第四章 多边形填充

计算机图形学

本章主要内容：

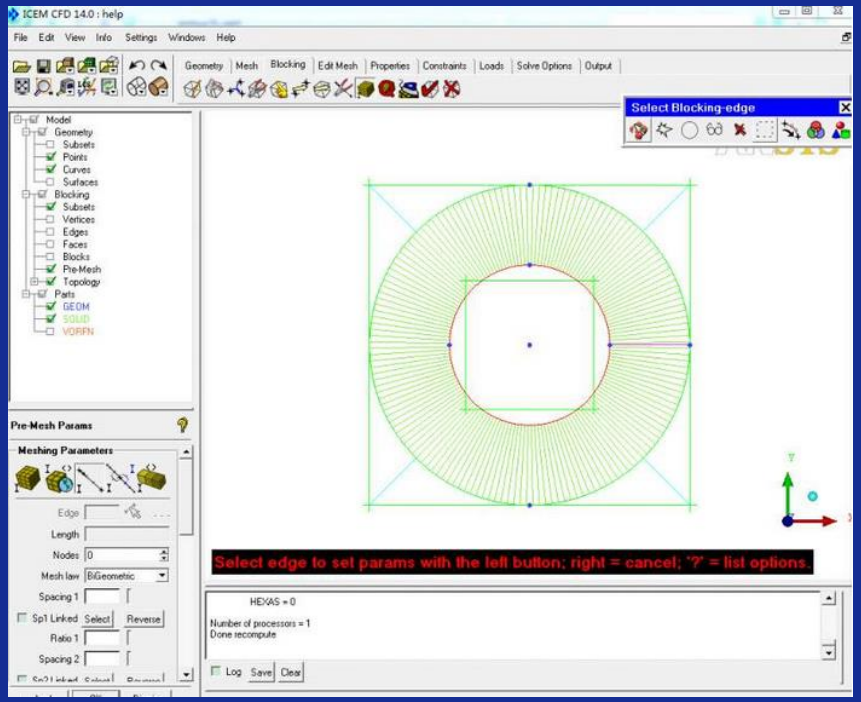
- 4.1 多边形的扫描转换
- 4.2 有效边表填充算法
- 4.3 边缘填充算法
- 4.4 区域填充算法
- 4.5 本章小结
- 习题4

本章学习目标：

- 了解多边形转换的基本概念
- 熟练掌握多边形有效边表填充算法
- 掌握多边形边缘填充算法
- 熟练掌握区域四邻接点和八邻接点区域填充算法
- 掌握区域扫描线种子填充算法

4.1多边形的扫描转换

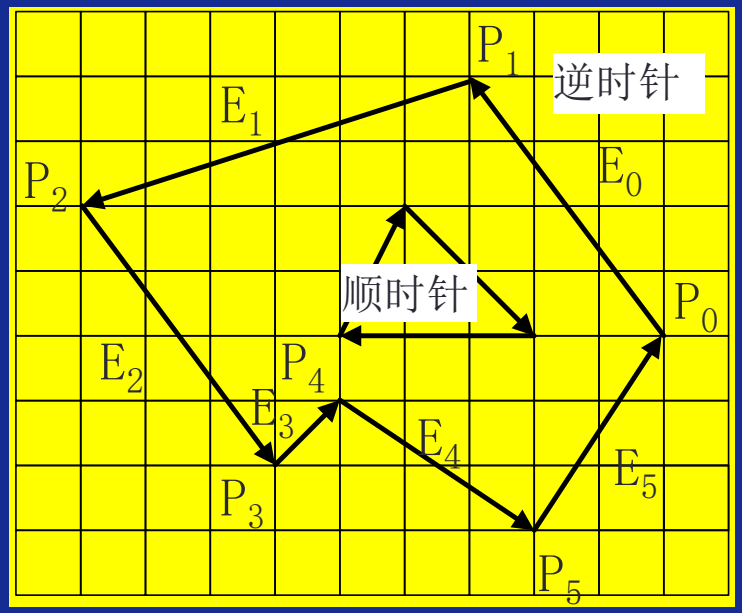
- 多边形的填充算法
- 图形边界像素的处理原则



圆环的网格模型
ICEM

圆环的光照模型

4.1.1 多边形的定义

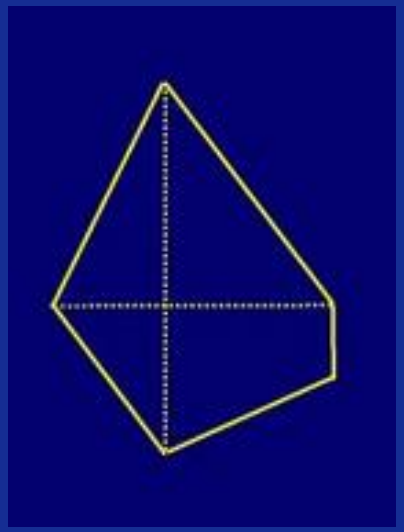


多段直线段彼此连接并且闭合
就构成了多边形

多边形由有序顶点的点集
$P_i(i=0,1,...n-1)$及
有向边的线集$E_i(i=0,1,...n-1)$定义，
n为多边形的顶点数或边数，
且$E_{i=}\ P_i\ P_{i+1}$ (i=0,1,...n-1)

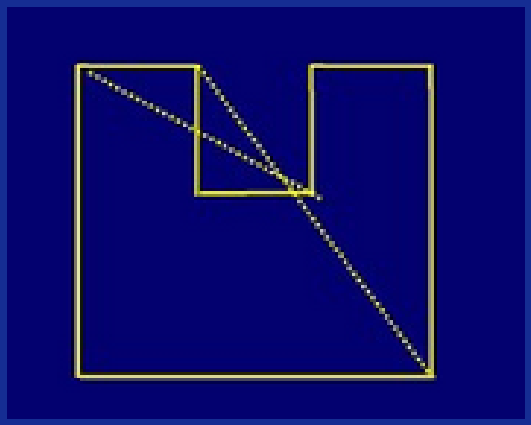
$P_{n=}\ P_0$保证了多边形
的封闭性

多边形的分类



凸多边形

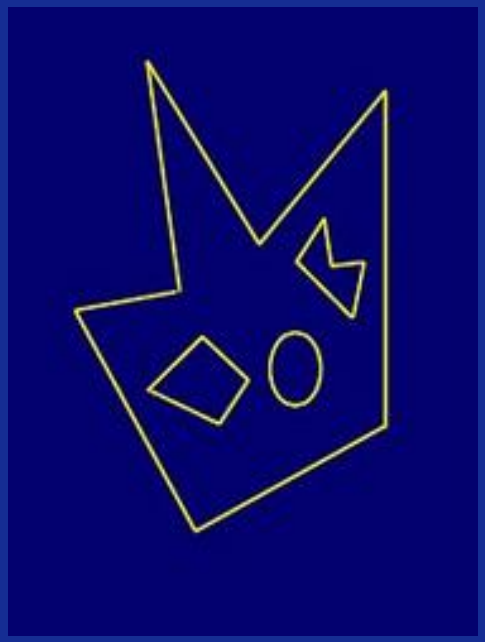
任意两顶点的连线均在多边形内，凸点对应的内角小于180°



凹多边形

任意两顶点的连线可能不在多边形内，
凹点对应的内角大于180°

多边形的分类

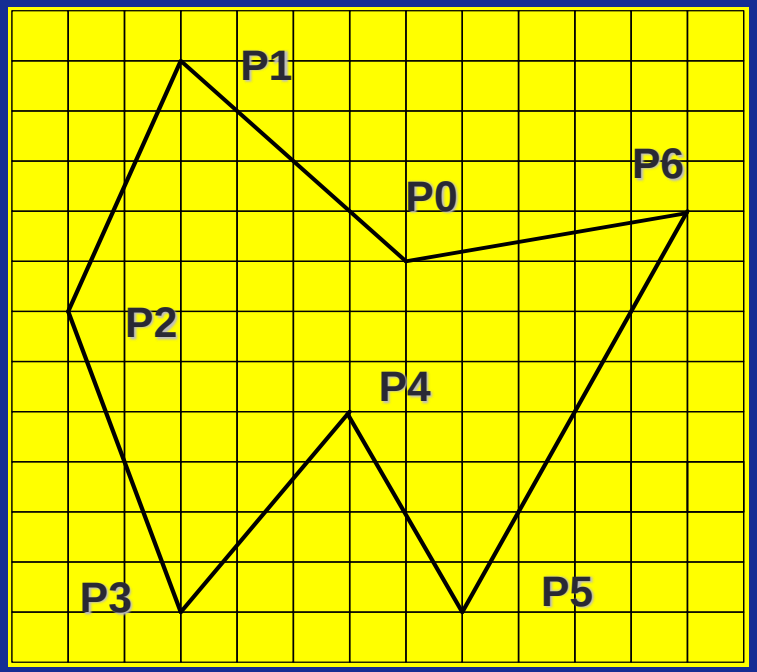


含内环的
多边形

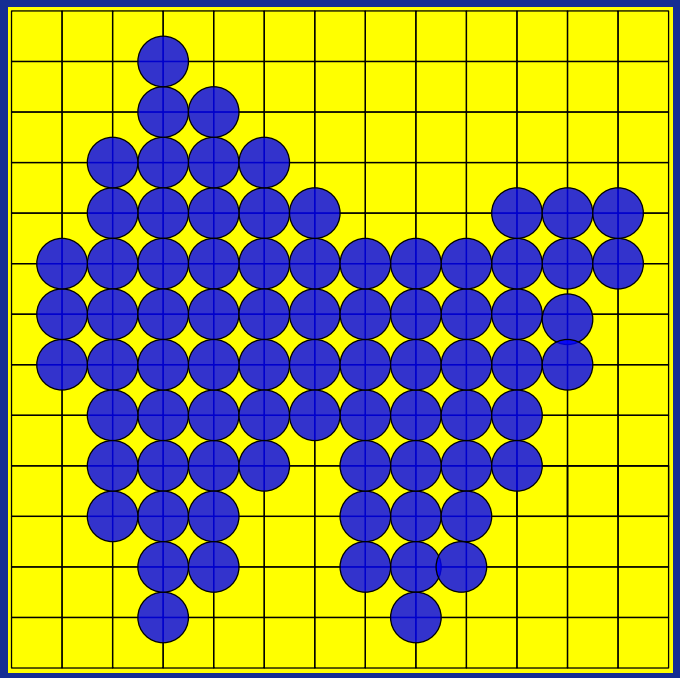
如果规定：每条有向边的左侧为其内部实面积区域。则当观察者沿着边界行走时，内部区域总在其左侧，也就是说多边形外轮廓线的环行方向为逆时针，内轮廓线的环行方向为顺时针。这种定义了环行方向的多边形称为环。

4.1.2 多边形的表示

- 多边形的表示有两种：顶点表示法和点阵表示法



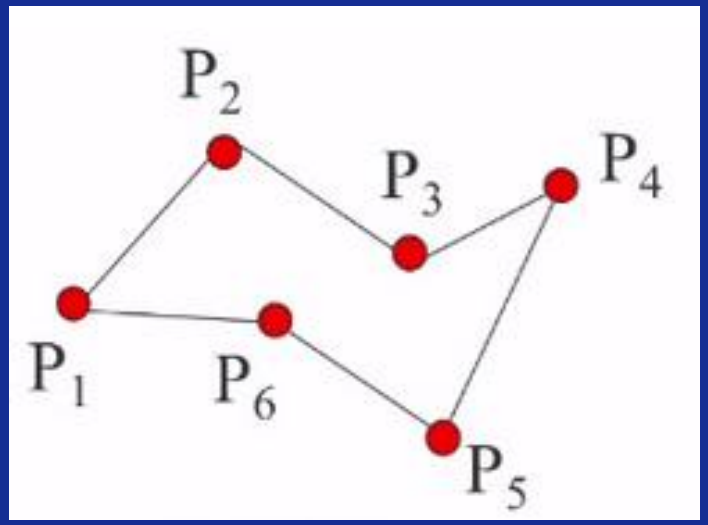
多边形的顶点表示法



多边形的点阵表示法

4.1.2 多边形的表示

■顶点表示：用多边形的顶点序列来表示多边形。



优点：

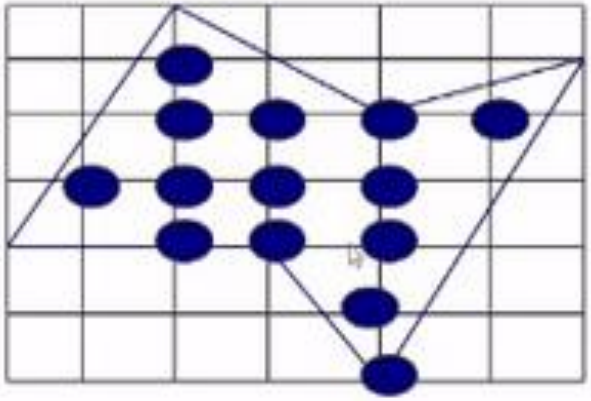
这种表示直观、几何意义强。
占内存少、易于进行几何变化

缺点：

由于它没有明确指出哪些像素在多边形内，故不能直接用于面着色
需要对多边形进行扫描转换后才能逐条扫描填充

4.1.2 多边形的扫描转换与区域填充

- 点阵表示：用位于多边形内的像素几何来刻画多边形。



优点：便于帧缓冲器表示图形

缺点：这种表示丢失了许多几何信息（如边界、顶点等）

多边形的扫描转换

- 把多边形的顶点表示转换为点阵表示，也就是从多边形的给定边界出发，求出位于其内部的各个像素素，并给帧缓冲器内的各个对应元素设置相应的灰度和颜色，通常称这种转换为多边形的扫描转换。

- 即从多边形的顶点信息出发，求出位于多边形内部的各个像素点信息，并将其颜色信息写入帧缓冲的相应单元。

思考

- 第一个问题是如果知道边界，能否求出哪些像素在多边形内？

顶点表示 → 点阵表示

可以

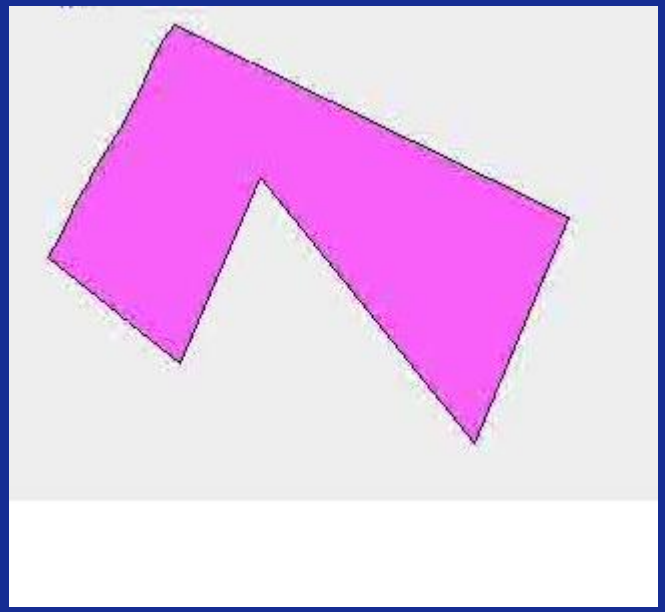
- 第二个问题是知道多边形内部的像素，反过来如何求出边界？

点阵表示 → 顶点表示

不可以

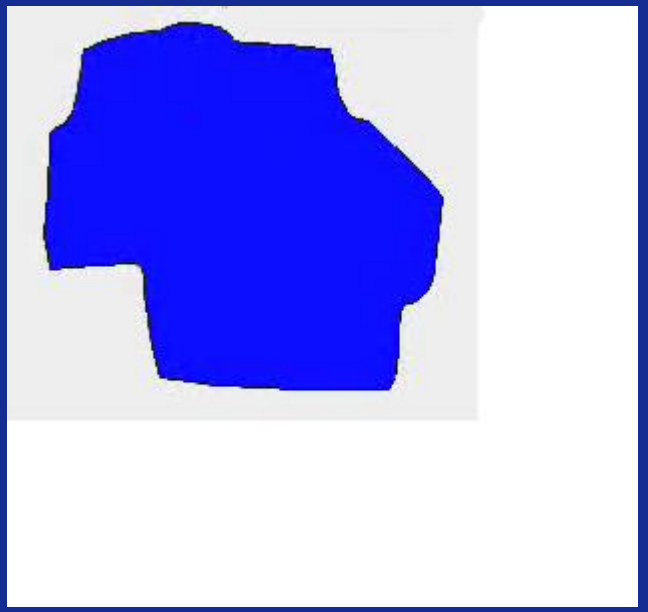
多边形的扫描转换

- 其中最具代表性的是：适应于顶点表示的扫描线类算法和适应于点阵表示的种子填充类算法。



•简单边界

•通过扫描线与边界交点确定填充区域



•复杂边界

• 从内部指定位置开始填充，递归填充直至边界

多边形的扫描转换

- 需解决的问题:知道多边形的边界，如何找到多边形内部的点，并将多边形内部填上颜色。

一般步骤:

- 确定那些像素位于填充区域内(顶点表示转换为点阵表示);
- 确定以什么颜色填充这些像素(区域填充);

4.1.5 填充区域

区域

相互连通的一组像素的集合。区域通常由一个封闭的边界来定义，处于一个封闭边界内的所有像素构成一个区域。

填充算法通常只对区域内部进行填充，区域内的所有像素着同一填充色，区域的边界色和填充色一般不同。

4.1.3 多边形的着色模式

- 平面着色模式（Flat Shading Mode）

  使用多边形第一个顶点的颜色填充，多边形内部具有单一颜色

- 光滑着色模式（ Smooth Shading Mode）

  多边形各个顶点的颜色不同，多边形内部的颜色由各个顶点的颜色进行双线性插值得到。

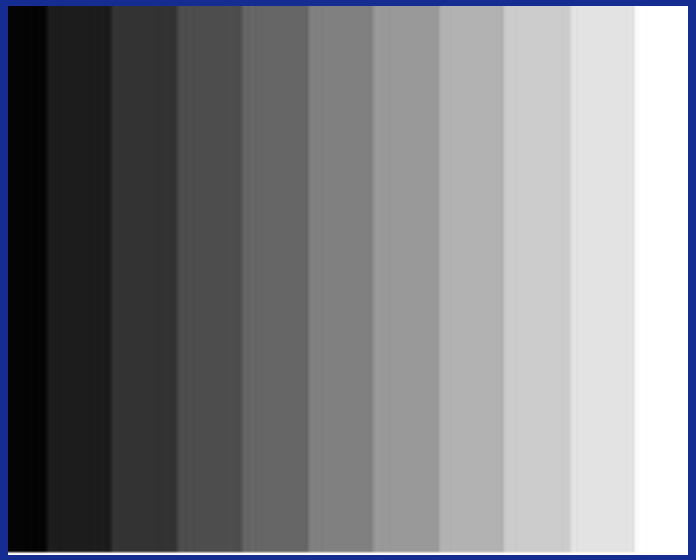
平面着色

光滑着色

4.1.3 多边形的着色模式

■马赫带效应（Mach Band Effect）

当观察两块亮度不同的区域时，边界处亮度对比度加强，使轮廓表现得非常明显。在真实感图形绘制时，应避免这种情况的产生。



马赫带

■意义：能使我们看到的轮廓更加清晰

4.1.4 填充多边形

- 填充多边形最简单的方法是检查光栅上的每一个像素是否位于多边形之内

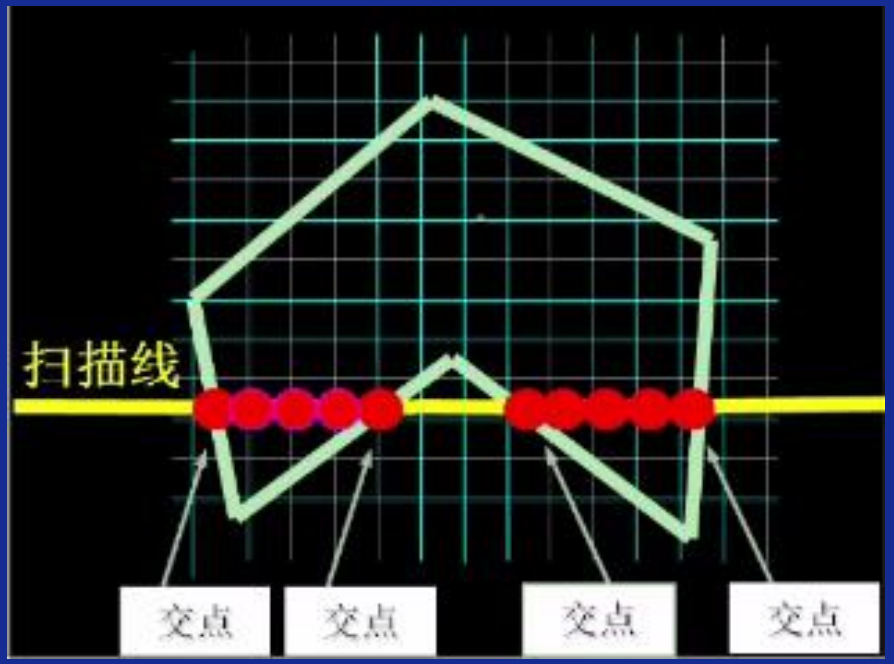
- 扫描线：屏幕上平行于X坐标轴且相互间距为一个像素的一系列直线

- 扫描线算法：用水平扫描线（y坐标）从上到下（或从下到上）扫描由多条首尾相连的线段构成的多边形

4.1.4 填充多边形

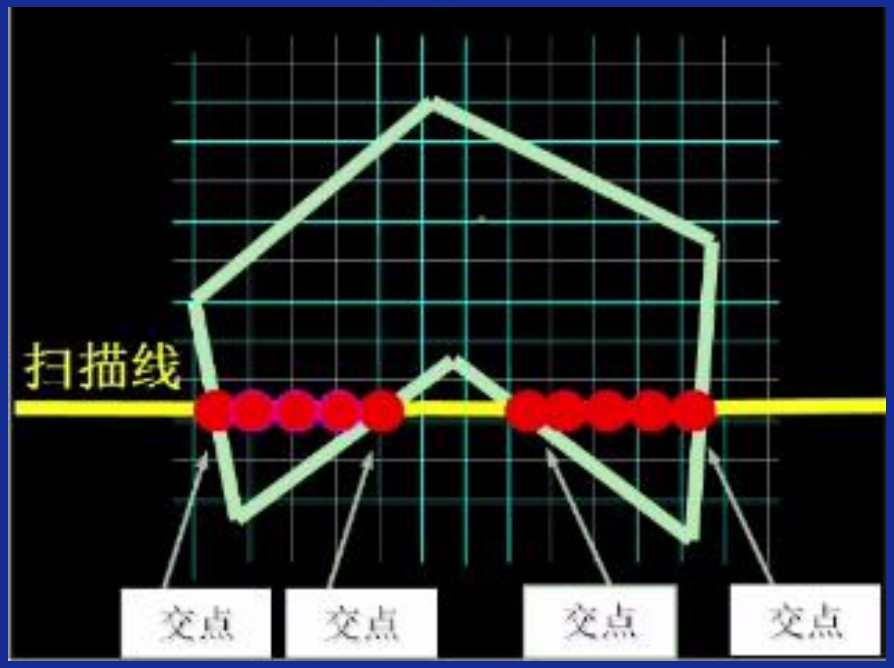
- 扫描线算法
- 第一步确定y的坐标范围：先确定多边形覆盖的扫描线条数

即y = y_{min}~y_{max}



4.1.4 填充多边形

- 第二步确定x的坐标范围：计算扫描线与多边形的相交区间,每根扫描线与多边形的某些边产生一系列交点,如果能判断该相交区间在多边形内部则将其内的像素绘制为指定的颜色。



- 在扫描线从多边形顶点的最小值y_{min}向多边形顶点的最大值y_{max}移动的过程中重复上述工作，就能完成多边形的填充。

4.1.4 填充多边形

- 对于一条扫描线，多边形的填充过程分为如下4个步骤：

（1）求交。计算扫描线与多边形各边的交点。

（2）排序。把所有交点按x值的递增顺序排序。

（3）配对。将第一个与第二个、第三个与第四个等交点配对，每对交点代表扫描线与多边形的一个相交区间。

（4）填色。把相交区间的像素置成多边形的颜色，把相交区域外的像素置成背景色。

4.1.4 填充多边形

■扫描线算法缺点

在处理每条扫描线时，需要与多边形的所有边求交，处理效率很低。

这是因为一条扫描线往往只与少数几条边相交，不相交的边占大多数。

通过建立有效边表和边表（桶表）来实现。即有效边表填充算法

4.2 有效边表算法

- 改进的扫描线算法就是有效边表填充算法：

按照扫描线从小到大的移动顺序，计算当前扫描线与有效边的交点，然后把这些交点按x值递增（从小到大）的顺序进行排序、配对，以确定填充区间，最后用指定颜色填充区间内的所有像素，即完成填充工作。

- 对于有效边表填充算法通过维护有效边表和边表，避开了扫描线与多边形所有边求交的复杂运算。该算法可以填充凸多边形、凹多边形和环。

4.2.4 边表

■有效边表（ Active Edge Tabel, AET ）

多边形内与当前扫描线相交的边称为有效边。

给出了扫描线与有效边的交点坐标的计算方法，但没有给出新边出现的位置信息。为了确定在哪条扫描线上插入了新边就需要构造一个边表。

■边表（ Edge Tabel, ET ）

用以存放在该扫描线第一次出现的边。

若某边的较低端点为y_{min},则该边就放在扫描线y_{min}的新边表中。

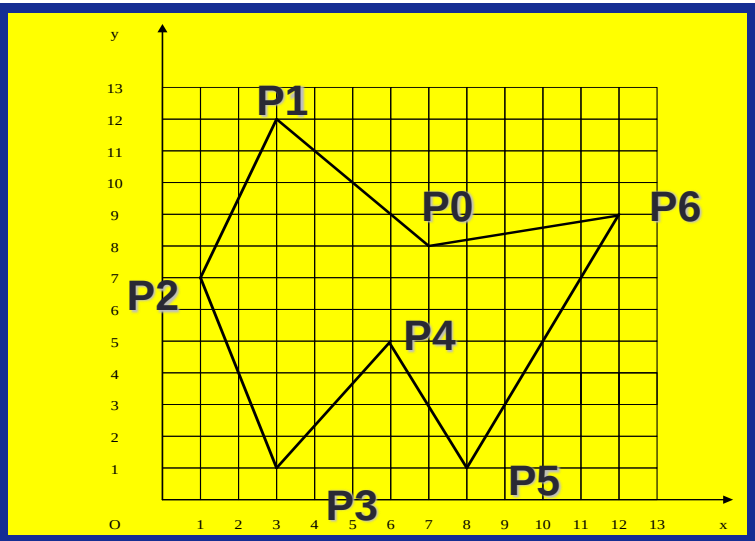
- 有效边表AET的意义(4.2.3)

在填充过程中，为每一条扫描线建立相应的有效边表，它表示出该扫描线要求交点的那些边，在实用中每一条边的AET表的信息与上一条边的AET表的信息有继承性，再结合ET表使得建立十分方便。

- 边表ET的意义(4.2.4)

是避免盲目求交。当处理一条扫描线时，为了求出它与多边形边的所有交点，必须将它与所有的边进行求交测试。而实际上只有某几条边与该扫描线有交点。边表正是用来排除不必要的求交测试的。即，它是为扫描线提供待加入的新边信息。

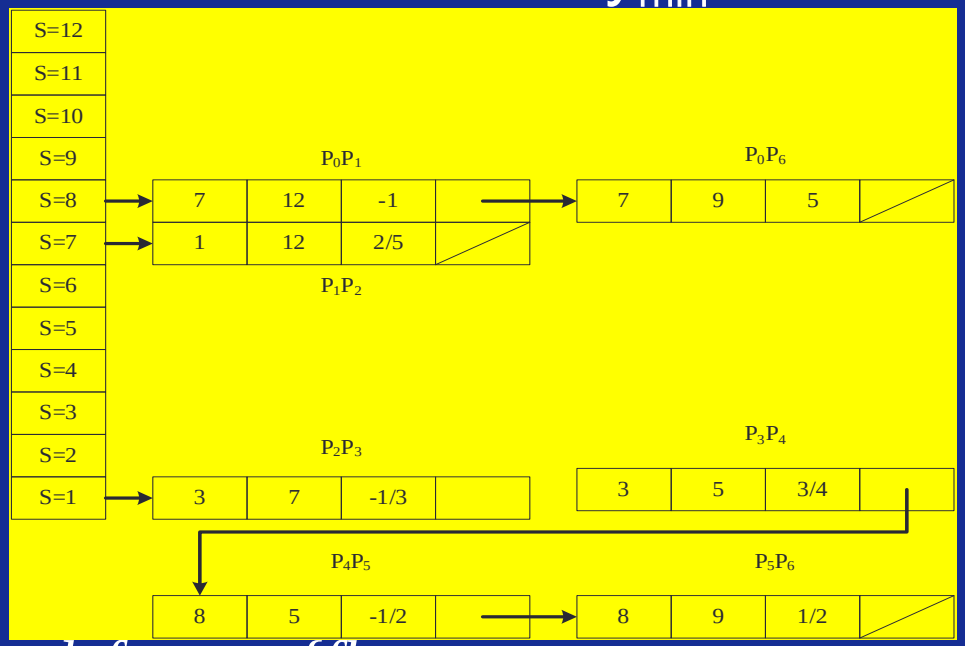
4.2.4 边表



- 顶点表示法为：P0（7，8），P1（3，12），P2（1，7），P3（3，1），P4（6，5），P5（8，1），P6（12，9）。
- 可以得出以下两点重要信息：
  - 每条边的最小x、y坐标和最大y坐标（最后一条扫描线）
  - 扫描线的最大值y_{max}=12(P1的纵坐标)最小值y_{min}=1（如P3的纵坐标）

边表数据结构

索引是边下端y_{min}=i




```
typedef struct {float x;
                int ymax;
                float  k;
               Edge *nextEdge;
               }Edge;
```

x:边下端的x坐标（最小x坐标）

ymax：该边所交的最高 扫描线号 y_{max}（最大y坐标）

1/k：该边斜率的倒数

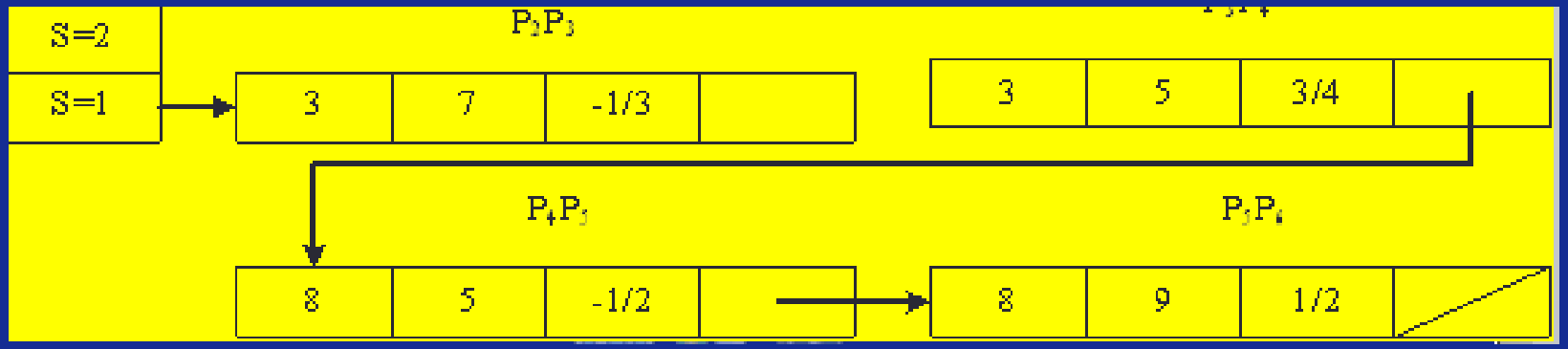
*nextEdge：指针

建立边表的步骤

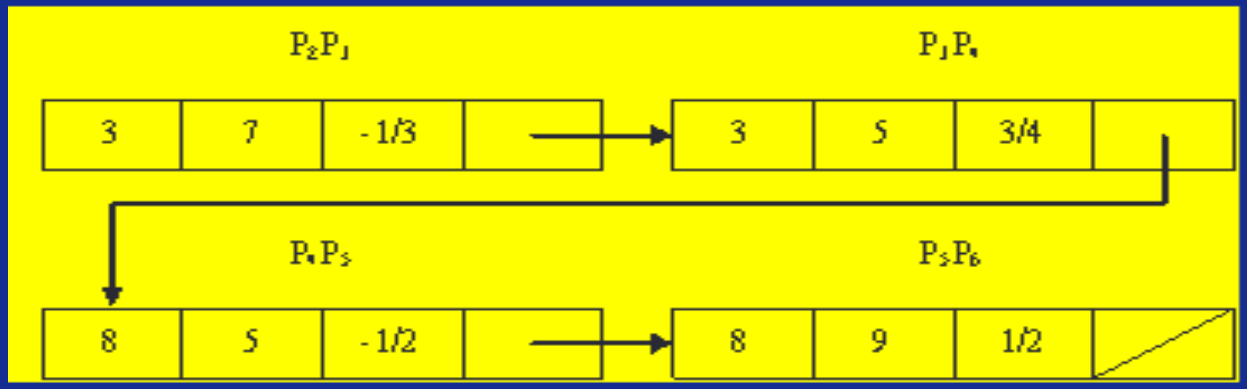
- 由所有顶点信息，得到扫描线的取值范围
- 由每条边的顶点信息，给出每条边y坐标的最小值，该值确定了存放在哪条扫描线的边表中；填入相应的数据结构结点信息；
- 扫描线的初始值$y_{min}=y_1$的结点信息作为有效边表的$y_{min}=y_1$ 的初始信息。
- 注意：边表中每条边只出现一次

边表与有效边表的初始对应

多边形的边表



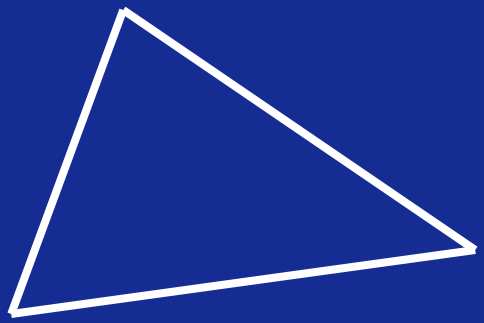
y=1有效边表

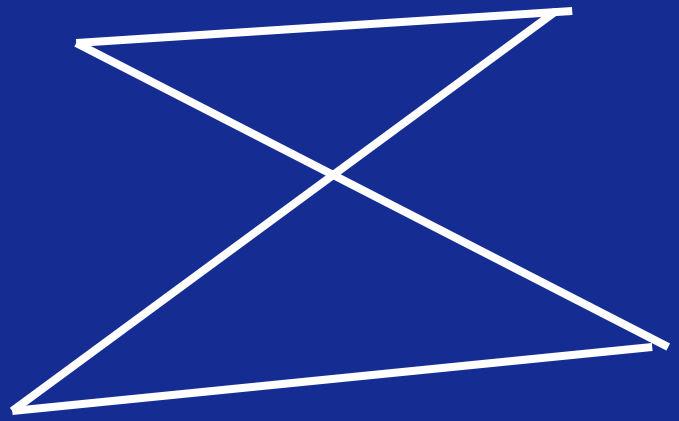


4.2 有效边表填充算法

有效边表填充算法

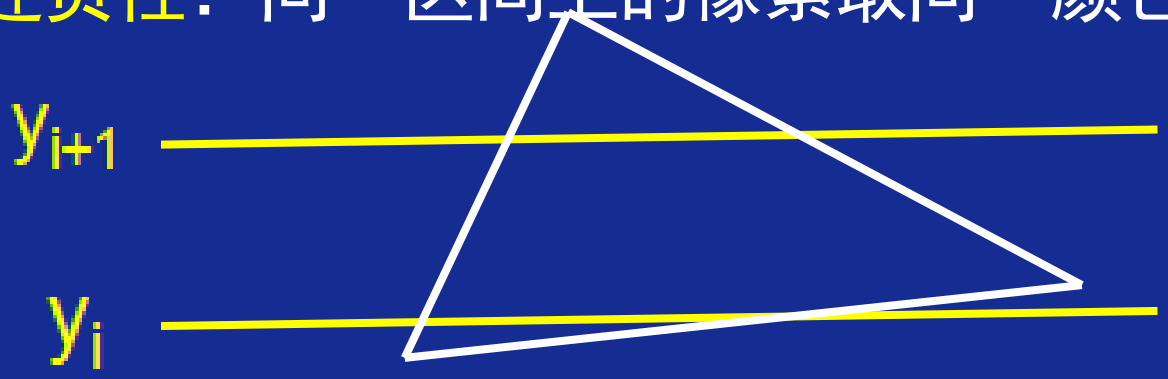
- 目标：利用相邻像素之间的连贯性，提高算法效率
- 处理对象：非自交多边形 （边与边之间除了顶点外无其它交点）





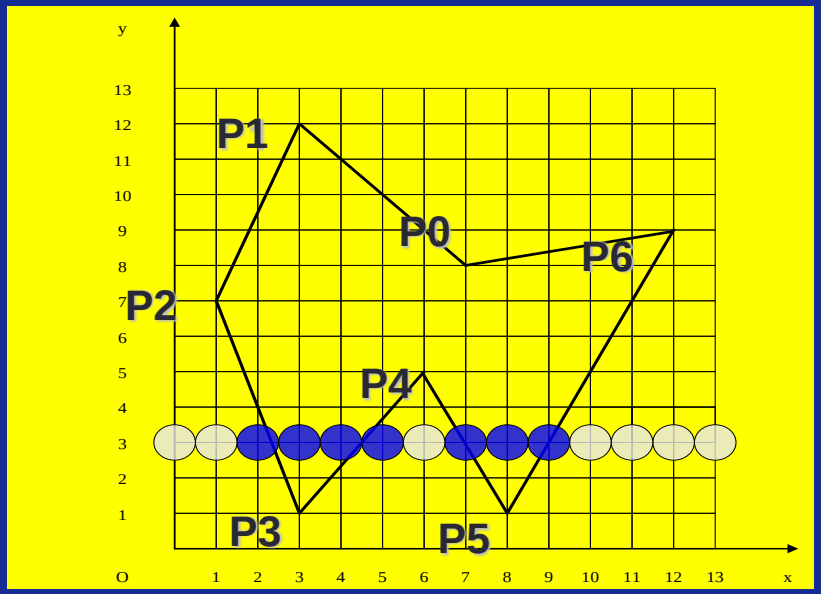
几个概念（有效边表填充算法的理论依据）

- **边的连贯性**：某条边与当前扫描线相交，也可能与
  - 下一条扫描线相交；
- **扫描线的连贯性**：当前扫描线与各边的交点顺序与
  - 下一条扫描线与各边的交点顺序可能相同或类似；
- **区间连贯性**：同一区间上的像素取同一颜色属性。



4.2.1 填充原理

- 多边形覆盖了12条扫描线。扫描线y = 3与多边形有4个交点（2.3，3），(4.5，3），（7，3）和（9，3）。对交点进行圆整处理后的结果为（2，3），(5，3），（7，3）和（9，3）。按x值递增的顺序对交点进行排序、配对后的填充区间为[2，5]和[7，9](偶数)，共有7个像素点需要填充为指定颜色。



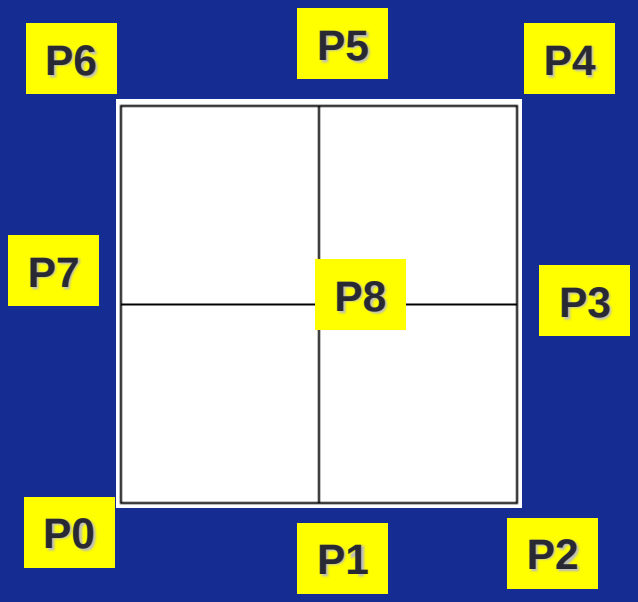
圆整规则

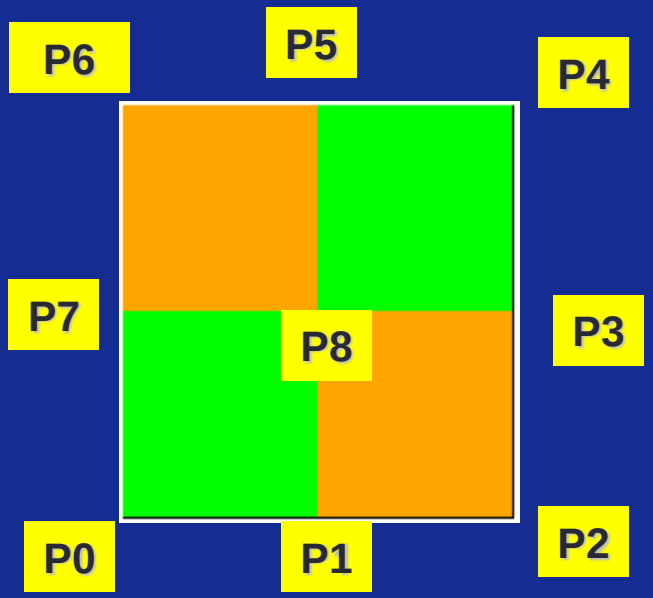
圆整规则

由于扫描线与边的交点一般为浮点型数值。假设一个区间的左交点为x_{left}，右交点为x_{right}，区间的左边界像素x_{first}和右边界像素 x_{last}通过对左交点x_{left}和右交点x_{right}进行四舍五入圆整得到的。

4.2.2 边界像素的处理原则

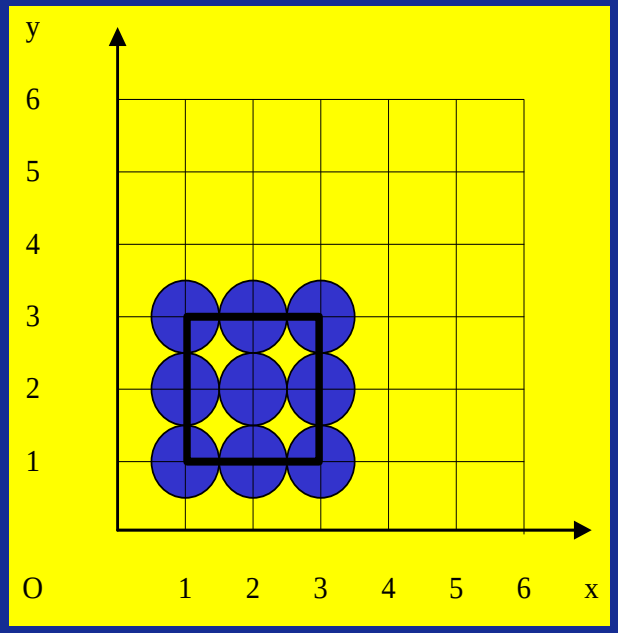
- 实际填充过程中，需要考虑到边界像素影响问题：四个小正方形的公共边为：P_1P_8、P_8P_5、P_7P_8和P_8P_3。

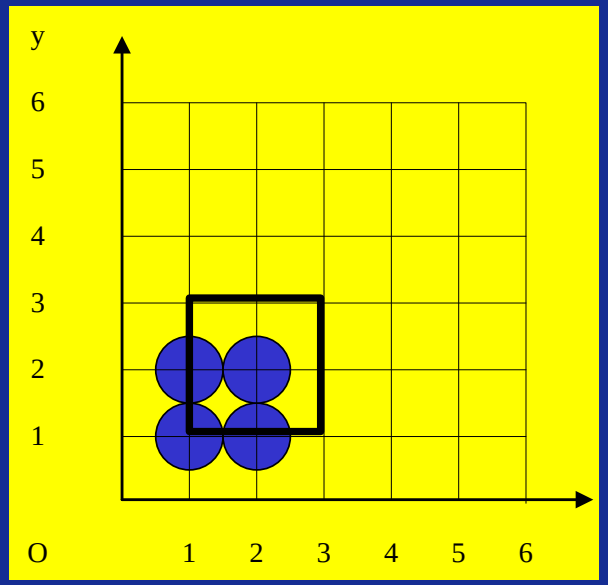




4.2.2 边界像素的处理原则

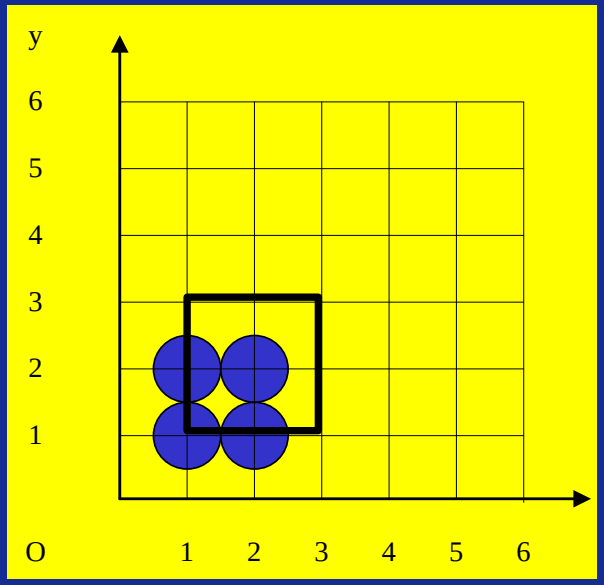
- 在实际填充过程中，也需要考虑到像素面积大小的影响问题：对左下角为（1，1），右上角为（3，3）的正方形进行填充时，若边界上的所有像素全部填充，就得到左图所示的结果。所填像素覆盖的面积为3×3个单位，而正方形的面积实际只有2×2个单位。





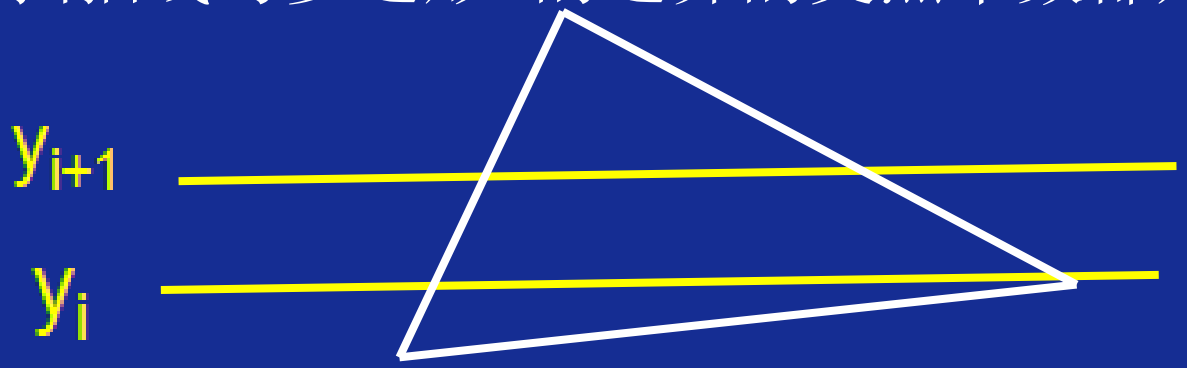
4.2.2 边界像素的处理原则

- 为了保解决这些问题，在多边形填充过程中，常采用“下闭上开”和“左闭右开”的原则对边界像素进行处理。即每个小正方形的右边界像素和上边界像素都没有填充，上面一行像素和右面一列像素没有填充。



4.2.2 顶点处理原则

■以上所述多边形的三种形式的连贯性都基于这样的几何事实：每一条扫描线与多边形P的边界的交点个数都是偶数。



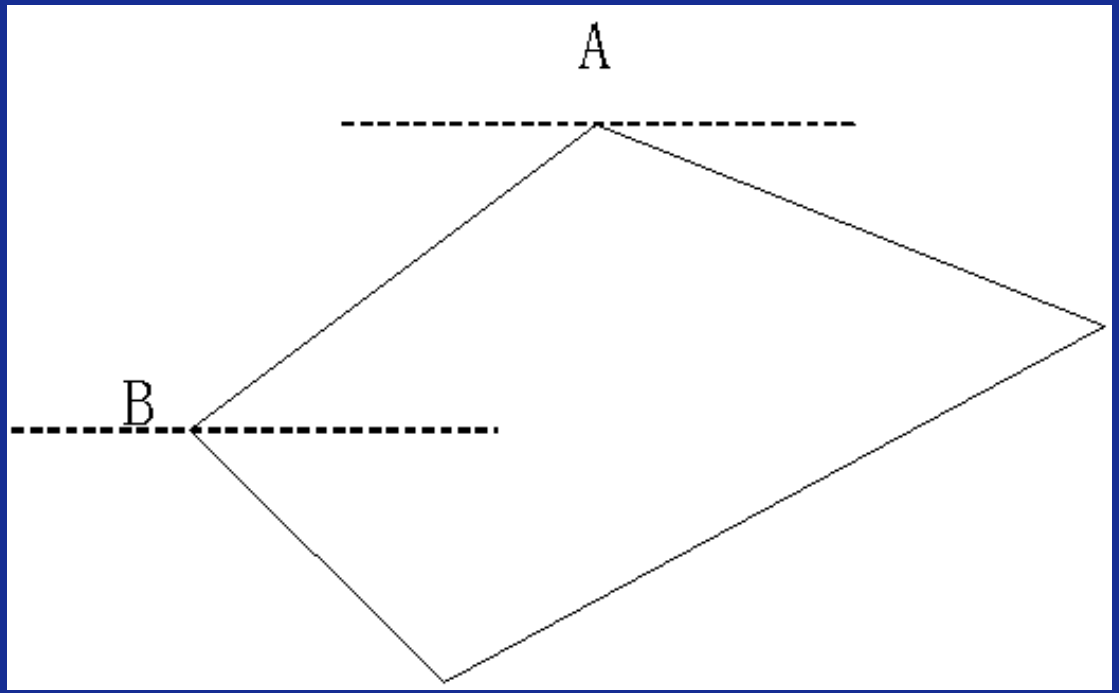
但是如果把每一奇点简单地计为一个交点或者简单地计为两个交点，都可能出现奇数个交点。

- 那么如果保证交点数为偶数呢？

■扫描线与多边形P的交点是线段的端点时，则称该交点为奇点。

4.2.2 奇点处理原则

- 若奇点做一个交点处理，则情况A，交点个数不是偶数。
- 若奇点做两个交点处理，则情况B，交点个数不是偶数。



4.2.2 奇点处理原则

- 多边形P的顶点可分为两类：极值奇点和非极值奇点。
- 如果$(y_{i-1} - y_i)(y_{i+1} - y_i) \geq 0$，则称顶点$P_i$为极值点；否则称$P_i$为非极值点。
- 简单判别法：
  - 从点出发画一条水平线，如果边都在同一侧，则是极值奇点。否则是非极值奇点。
  - 极值奇点不等于极大值或极小值。
- **规定：奇点是极值点时，该点按两个或0个交点计算，否则按一个交点计算。**

4.2.2 奇点处理原则

- 1.普通连接点的处理原则 ，共享顶点的两条边分别落在扫描线的两侧

如 P2是边P3P2的终点，同时也是边P2P1的起点。按照“下闭上开”的原则， P2 作为边P3P2的终点不填充，但作为边P2P1的起点要填充。

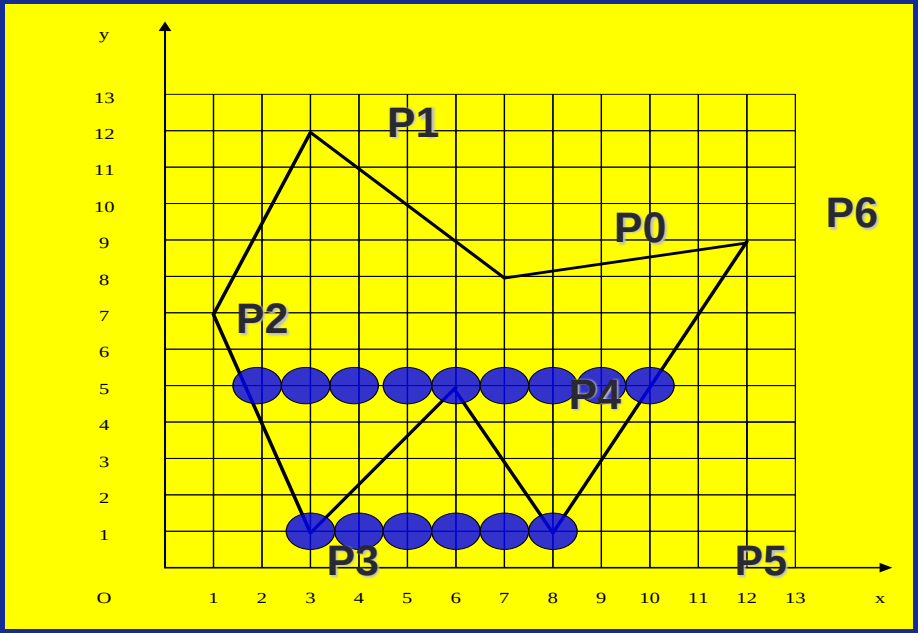
计一个交点P2。（根据共享顶点的两条边的另一端y的值大于扫描线y值的个数来将交点个数取为0、1、2）

4.2.2 奇点处理原则

■2.局部最低点的处理原则，共享顶点的两条边落在扫描线的上方

P0、P3和P5是局部最低点。如果处理不当，扫描线y＝1会填充区间（3，8），结果填充了P3和P5点之间的像素。

P0点、P3和P5计2个交点；



4.2.2 奇点处理原则

■3.局部最高点的处理原则

P4点是局部最高点。计0个交点。但局部最高点仍会被其他情况填充

P1、P6点不被填充（“下闭上开”）

■扫描线与边界重合

最下边界填充、最上边界不填充、内边界计一个交点

4.2 有效边表算法

- 改进的扫描线算法就是有效边表填充算法：

按照扫描线从小到大的移动顺序，计算当前扫描线与有效边的交点，然后把这些交点按x值递增（从小到大）的顺序进行排序、配对，以确定填充区间，最后用指定颜色填充区间内的所有像素，即完成填充工作。

- 对于有效边表填充算法通过维护有效边表和边表，避开了扫描线与多边形所有边求交的复杂运算。该算法可以填充凸多边形、凹多边形和环。

4.2.3 有效边与有效边表

■有效边 (Active Edge, AE)

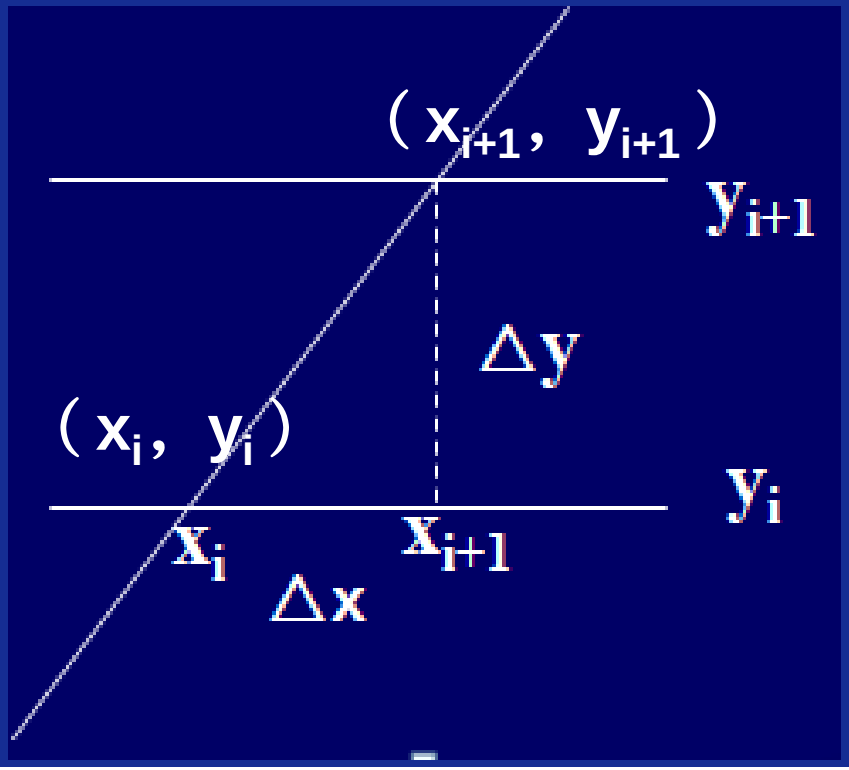
多边形内与当前扫描线相交的边称为有效边。

在处理一条扫描线时仅对有效边进行求交运算，可以避免与多边形的所有边求交，提高了算法效率。

有效边上的扫描线由起点到终点每次加1，即像素点的y坐标为：y = y+1，x坐标的变化可以按如下方法推导。

数据结构

- 为了提高速度，假定当前扫描线与多边形某一条边的交点的x坐标为xi，则下一条扫描线与该边的交点不要重新计算，而是通过增加一个增量△x来获得：利用两条扫描线间隔为1的特性，可以简化边与扫描线交点的计算。如下图所示：



设边AB的斜率为k，若其与扫描线y_i的交点横坐标为x_i，则与扫描线y_{i+1}的交点的横坐标为：$x_{i+1}= x_i +1/k$

$y_{i+1} =k\ x_{i+1} +b$

$x_{i+1}= 1/k（\ y_{i+1} -b）$

$= 1/k（\ y_i +1 -b）$

$= 1/k（\ k\ x_i +b+1 -b）$

$=x_i +1/k$

4.2.3 有效边与有效边表

- 有效边表 (Active Edge Table, AET)

把有效边按照与扫描线交点x 坐标递增的顺序存放在一个链表中，成为有效边表。有效边表利用了边的连贯性：

- 与扫描线y_i相交的边，大多与扫描线y_{i+1} 相交
- 从一条扫描线到下一条扫描线，交点x的增量相等。

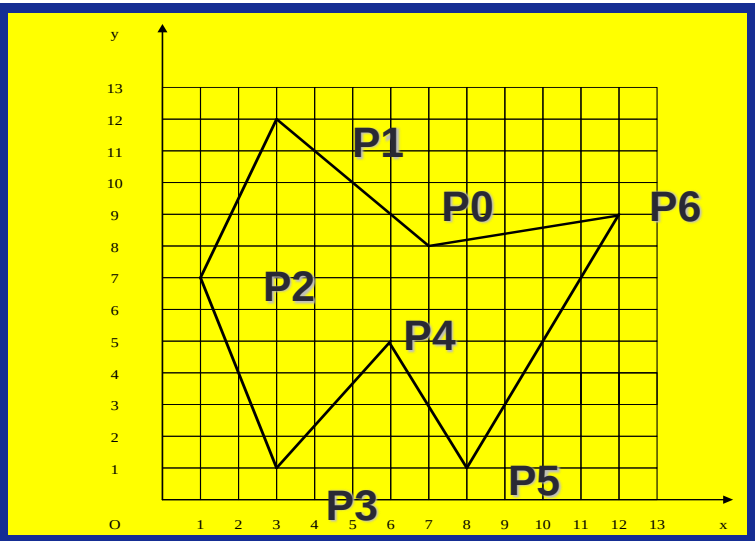
结点如下图

x	y_{max}	1/k	next

x为当前扫描线与有效边的交点横坐标；

y_{max} 为边所在的扫描线的最大值,用于判断该边何时扫描完毕后被抛弃成为无效边。

4.2.3 有效边表



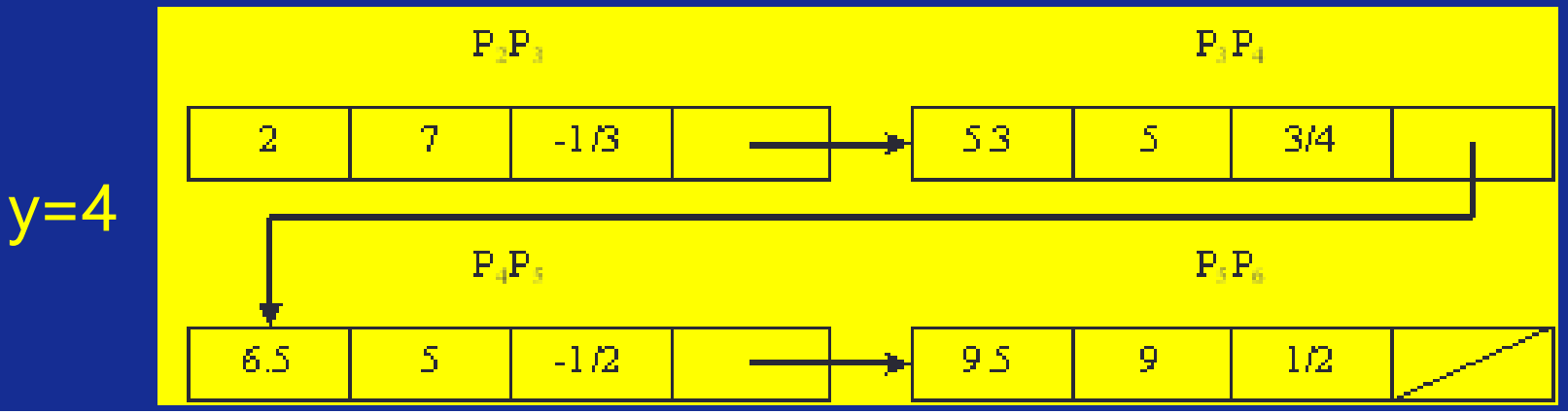
- 顶点表示法为：P0（7，8），P1（3，12），P2（1，7），P3（3，1），P4（6，5），P5（8，1），P6（12，9）。
- 可以得出以下两点重要信息：
  - 每条直线段（边）的表达式
  - 扫描线的最大值y_{max}=12(P1的纵坐标)最小值y_{min}=1（P3的纵坐标）

4.2.3 有效边表

- 1.由上一条扫描线的有效边表得到当前有效边表

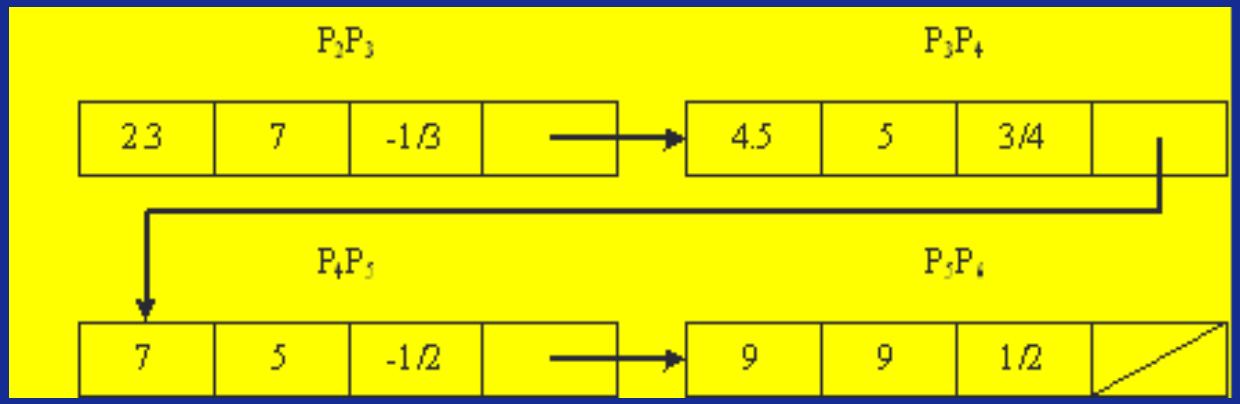
如扫描线y=4 的有效边表由扫描线y=3 的有效边表求出；

- 2.由上一条扫描线的有效边表的经过有效边的最大值扫描线判断是否删除该边，使其称为无效边；
- 3.数据结构中的计算公式得到与有效边表交点坐标值；
- 4.根据边表判断是否加入新边；



4.2.3 有效边表

y=3



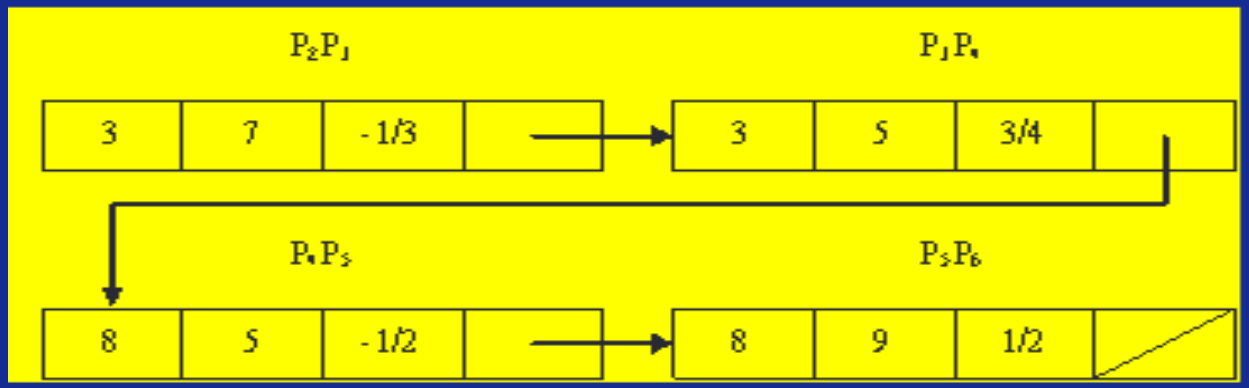
- 推导出的信息：
  - 对于边P2P3，由结点可知，$y_{max}=7$，则下一条扫描线y=4必定与其相交，且交点坐标为$x_4=x_3+1/k=7/3-1/3=2$ $y_4=4$

利用**边的连贯性**：边与当前扫描线相交，多数与下一条扫描线相交

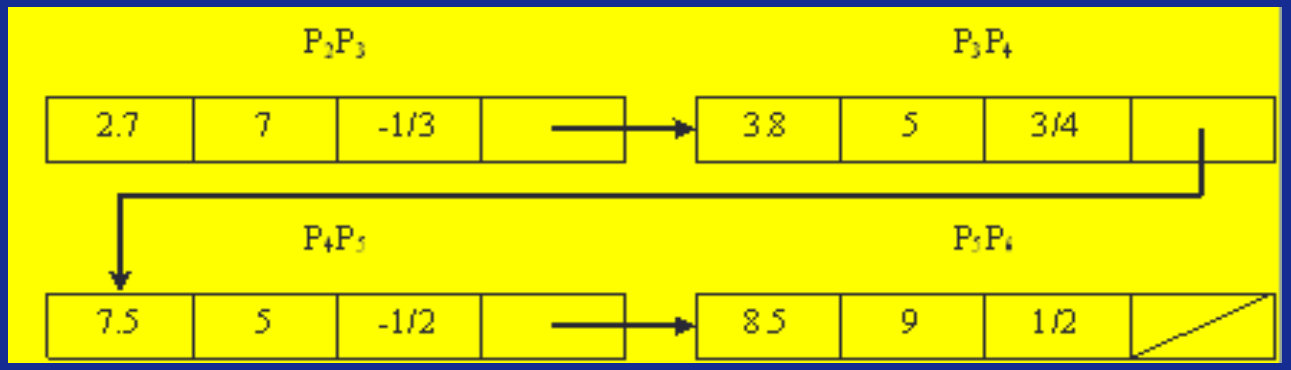
4.2.3 有效边表

- 建立上图多边形的全部有效边表（每一条扫描线对应一条有效边表）

y=1



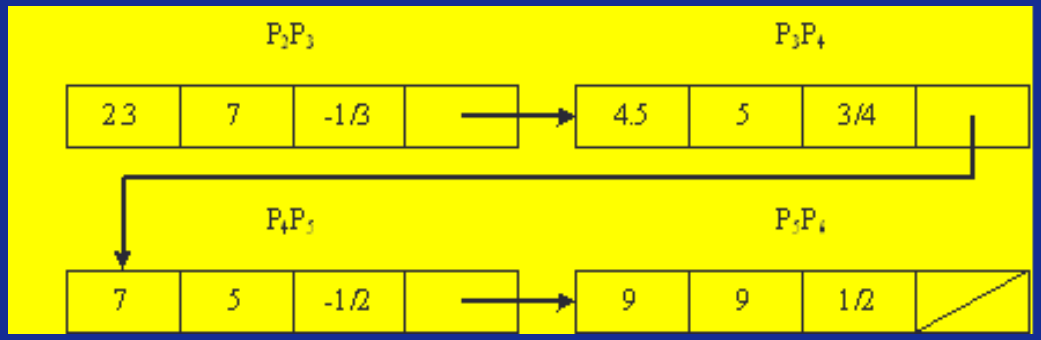
y=2



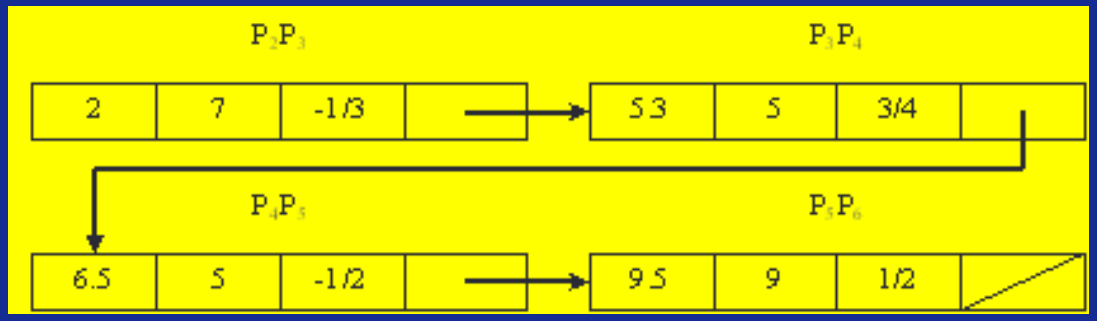
4.2.3 有效边表

- 建立上图多边形的全部有效边表（每一条扫描线对应一条有效边表）

y=3



y=4

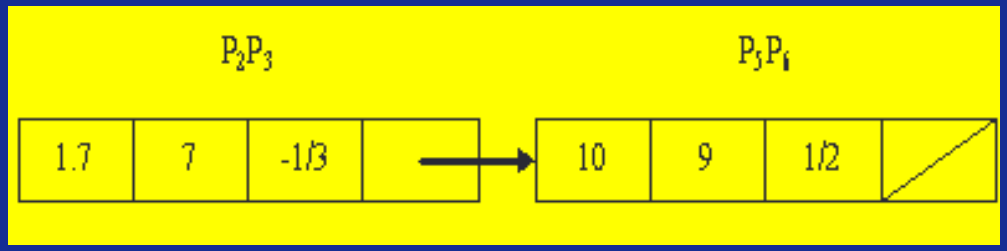


这条扫描线处理完毕后 对于P3P4和P4P5两条边，因为下一条扫描线y=5和ymax相等，根据“下闭上开”的原则予以删除。

4.2.3 有效边表

- 建立上图多边形的全部有效边表（每一条扫描线对应一条有效边表）

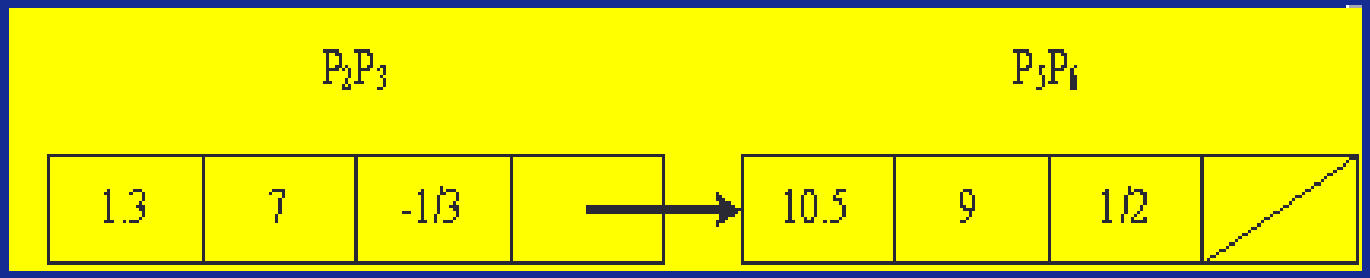
y=5



4.2.3 有效边表

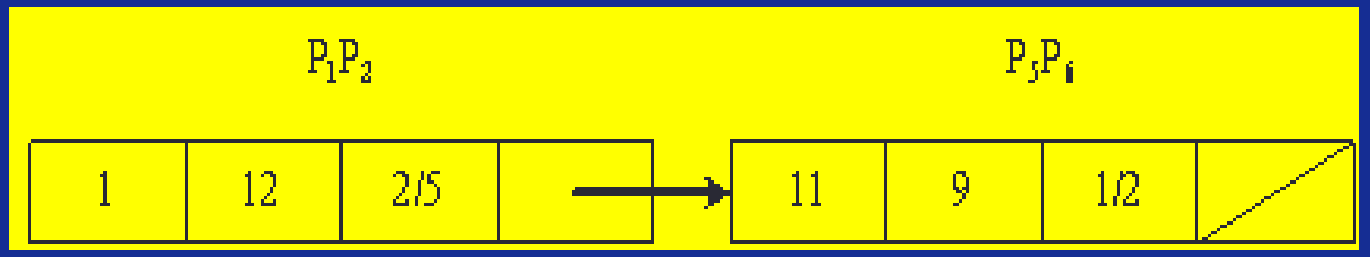
- 建立上图多边形的全部有效边表（每一条扫描线对应一条有效边表）

y=6



- 这条扫描线处理完毕后 对于P2P3边，因为下一条扫描线y=7和ymax相等，根据“下闭上开”的原则予以删除。 当y = 7时 ，添加上新边P1P2。

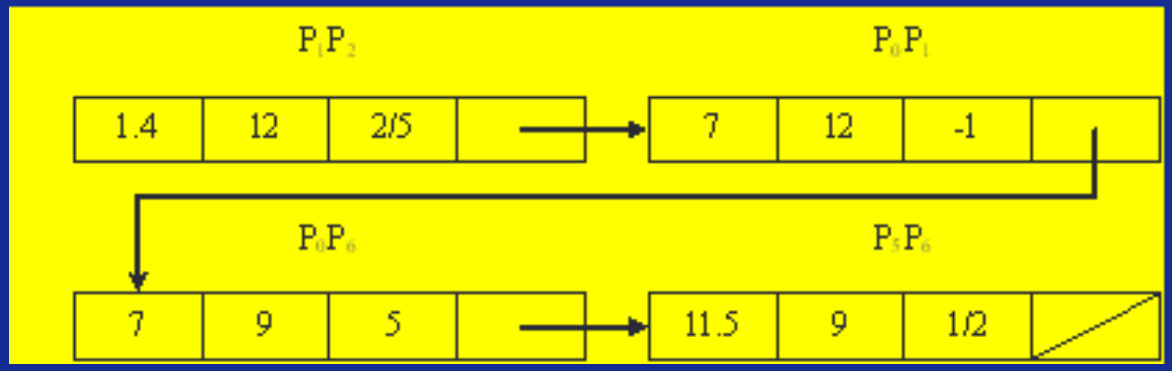
y=7



4.2.3 有效边表

- 当y = 8时 ，添加上新边P0P1和P0P6。

y=8

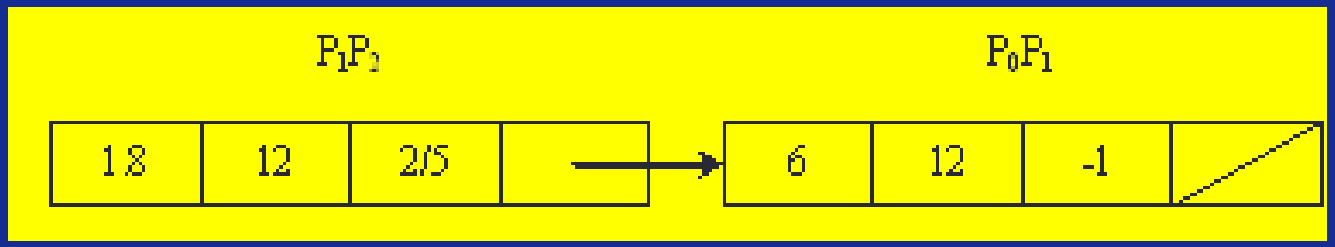


这条扫描线处理完毕后 对于P5P6和P0P6两条边，因为下一条扫描线y=9和ymax相等，根据"下闭上开"的原则予以删除。

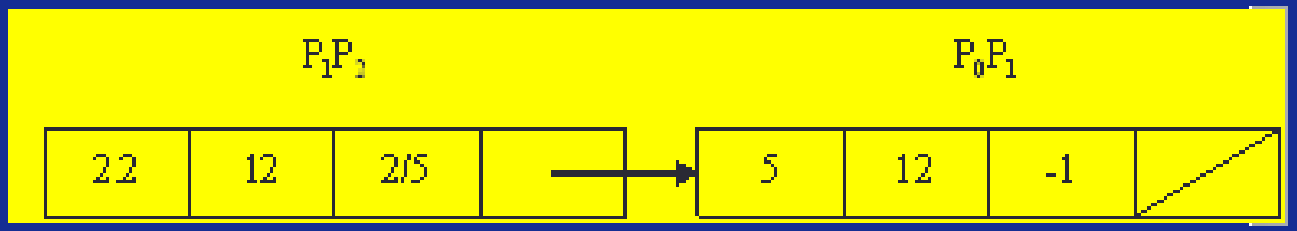
4.2.3 有效边表

- 建立上图多边形的全部有效边表（每一条扫描线对应一条有效边表）

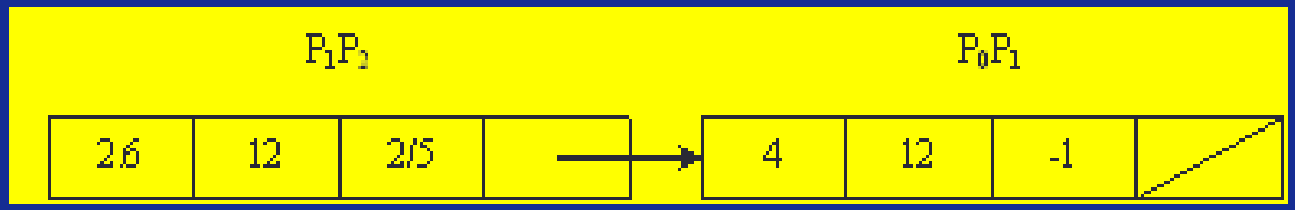
y=9



y=10



y=11



- y=11的扫描线处理完毕后 对于P1P2边和P0P1边，因为下一条扫描线y=12和ymax相等，根据“下闭上开”的原则予以删除。至此，全部有效边表已经给出。

4.2.4 边表

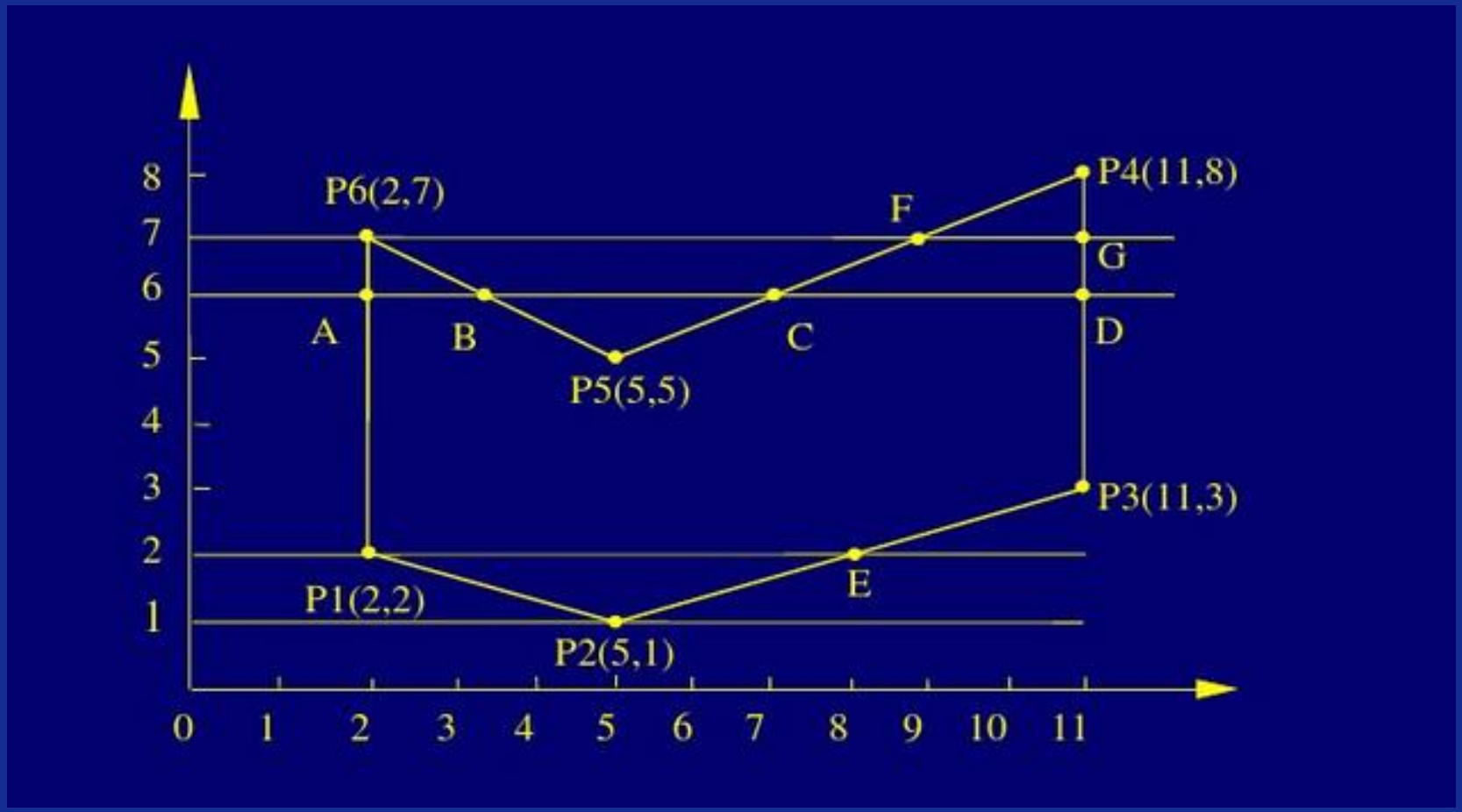
- 边表ET的意义：

是避免盲目求交。当处理一条扫描线时，为了求出它与多边形边的所有交点，必须将它与所有的边进行求交测试。而实际上只有某几条边与该扫描线有交点。新边表正是用来排除不必要的求交测试的。即，它是为扫描线提供待加入的新边信息。

- 有效边表AET的意义：

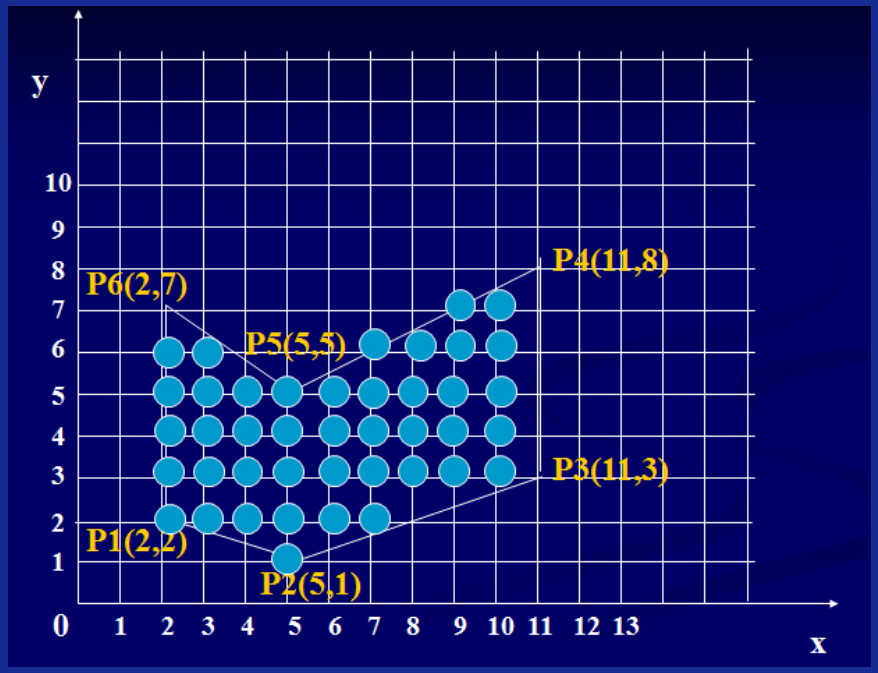
在填充过程中，为每一条扫描线建立相应的有效边表，它表示出该扫描线要求交点的那些边，在实用中每一条边的AET表的信息与上一条边的AET表的信息有继承性，再结合ET表使得建立十分方便。

有效边表扫描算法习题


P6(2,7)
P4(11,8)
F
G
A
B
C
D
P5(5,5)
P3(11,3)
E
P1(2,2)
P2(5,1)
0 1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8

有效边表扫描算法习题

- 1、建立边表：新边在扫描线上出现的位置
- 2、建立有效边表：扫描线与每条有效边的交点坐标
- 注意：判断交点为顶点时的特殊情况

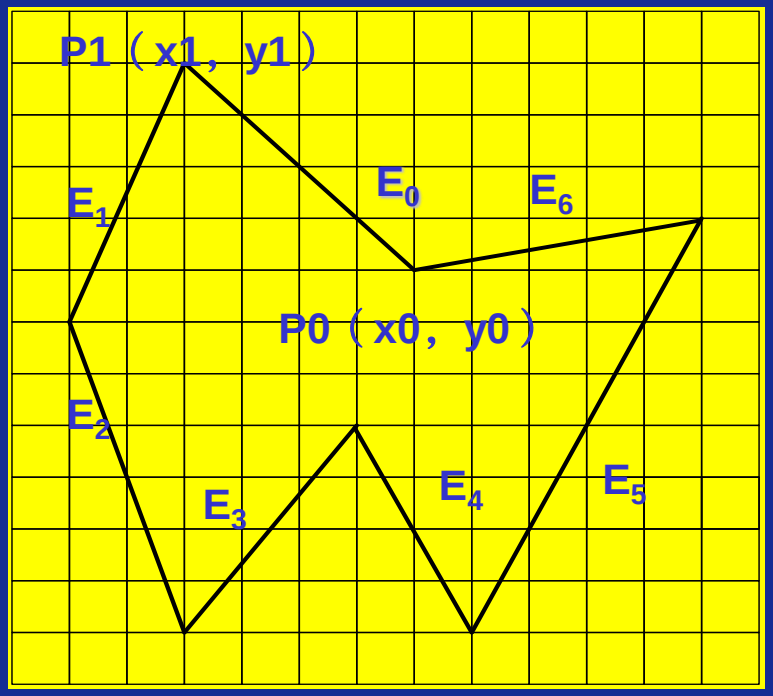


4.3 边缘填充算法

- 边缘填充算法是求出多边形的每条边与扫描线的交点，然后将交点右侧的所有像素颜色全部取为反色。按任意顺序处理完多边形的所有边后，就完成了多边形的填充任务。

4.3 边缘填充算法

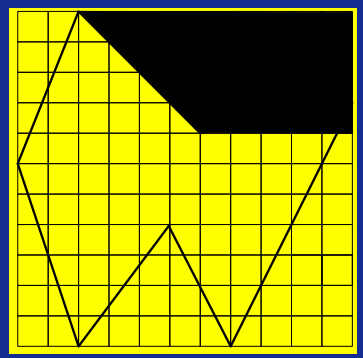
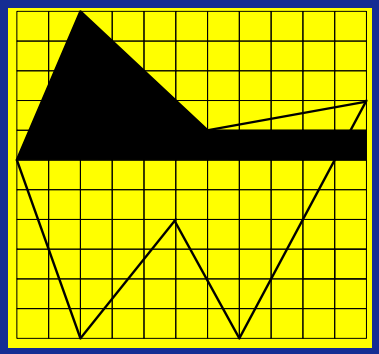
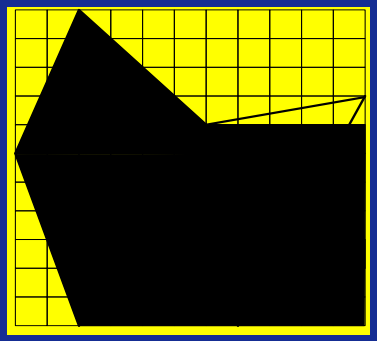
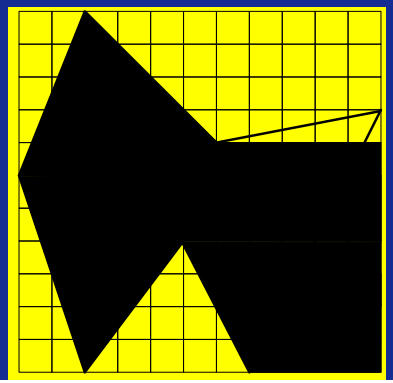
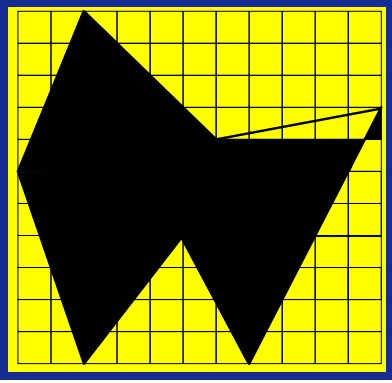
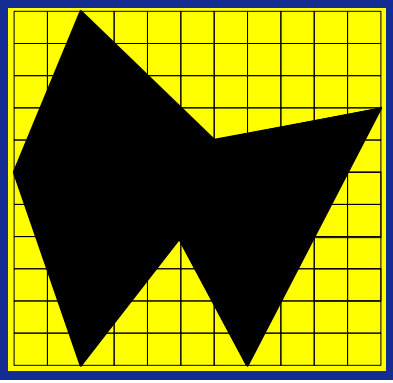
- 假定边的顺序为E0、E1、E2、E3、E4、E5、E6



4.3 边缘填充算法

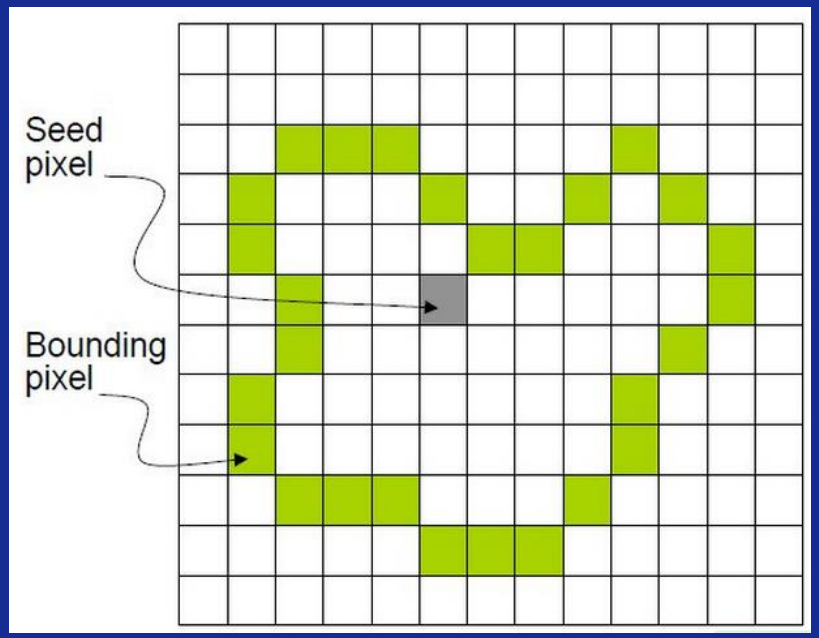
- 如果多边形内部像素被取反奇数次，则显示为填充色；

  如果多边形内部像素被取反偶数次，则保持为背景色。

4.4 区域填充算法

- 区域填充将指定的颜色从种子点扩展到整个区域的过程。能对具有任意复杂边界的区域进行填充
  - (1)从区域内部一个像素点开始,称为种子点(该点已知)
  - (2)判断这个像素是否是一个边界像素点或者已经被填充了
  - (3)如果都不是，就把它填充，然后开始设置邻居像素点。



4.4 区域填充算法

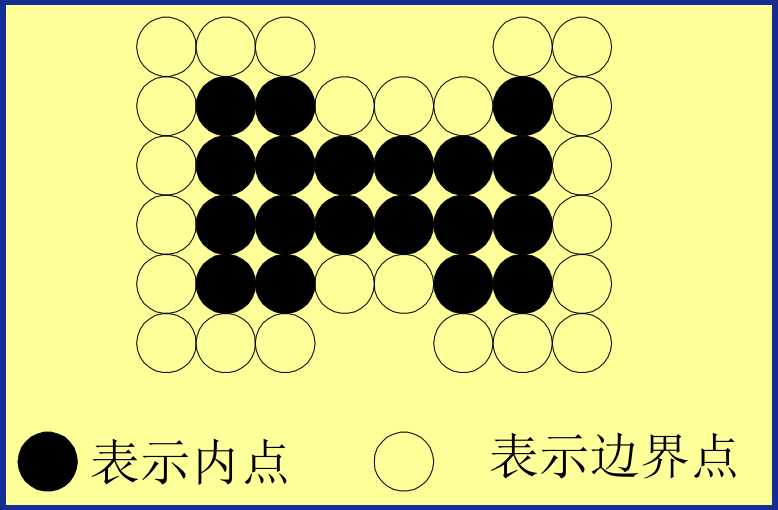
- 区域指已经表示成点阵形式的填充图形,它是像素的集合;区域指可采用内点表示和边界表示两种形式;

内点表示

- 枚举出区域内部的所有像素
- 内部的所有像素着同一个颜色
- 边界像素着与内部像素不同的颜色

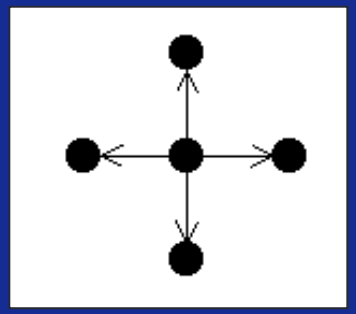
边界表示

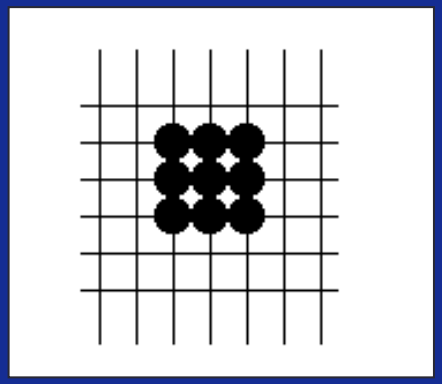
- 枚举出边界上所有的像素
- 边界上的所有像素着同一颜色
- 内部像素着与边界像素不同的颜色



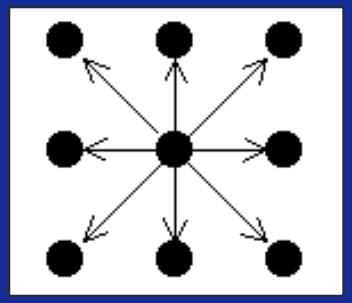
4.4.2 四邻接点和八邻接点

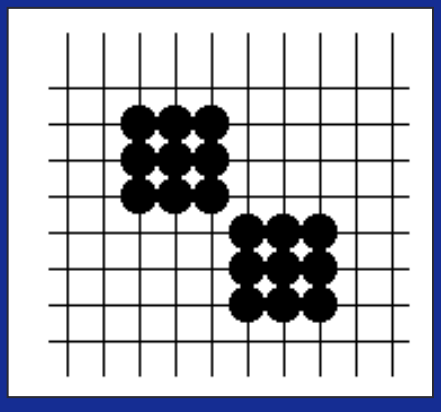
- 四邻接点





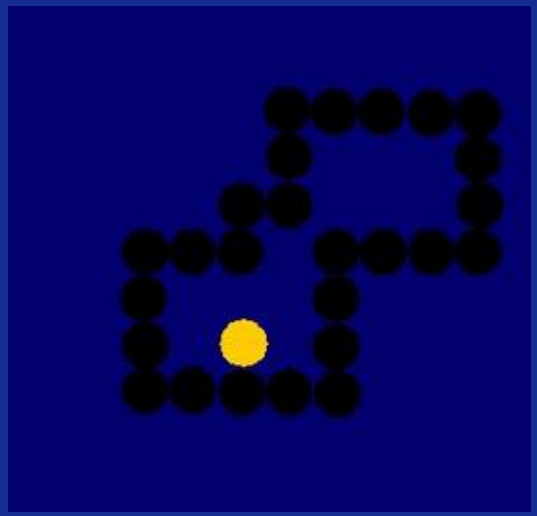
- 八邻接点

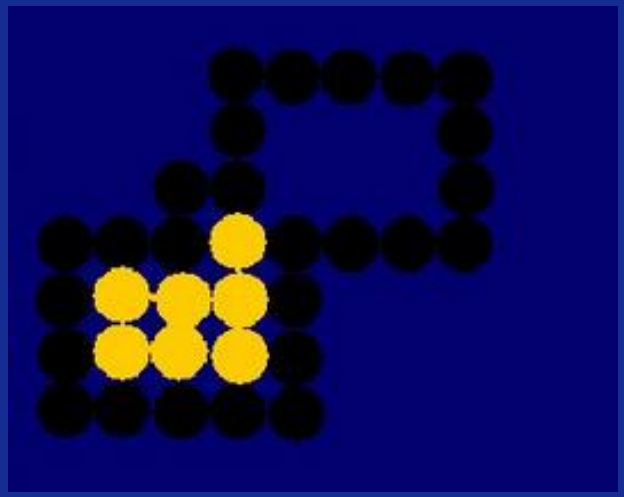




4.4.3 四连通域与八连通域

- 区域填充算法要求区域是连通的，这样才能将种子像素的颜色扩展到多边形区域内部的任意像素。





- 4连通区域指的是从区域上一点出发，可通过四个方向，即上、下、左、右移动的组合，在不越出区域的前提下，到达区域内的任意像素；

- 8连通区域指的是从区域内每一像素出发，可通过八个方向，即上、下、左、右、左上、右上、左下、右下这八个方向的移动的组合

4.4.4 种子填充算法

■算法定义

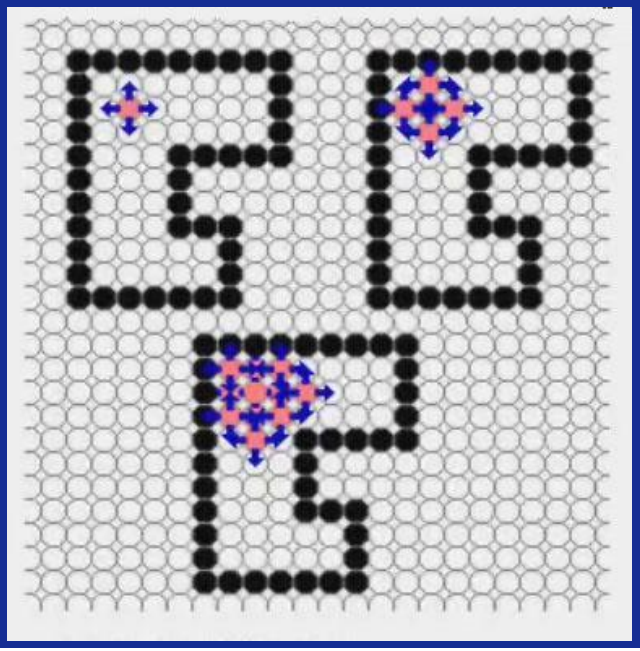
从种子像素点开始，使用四邻接点方式搜索下一个像素点的填充算法称为四邻接点种子填充算法（上、下、左、右）；

使用八邻接点方式搜索下一个像素点的填充算法称为八邻接点种子填充算法（上、下、左、右、左上、右上、左下、右下）

■四邻接点种子填充算法的缺点是不能通过狭窄区域，即不能填充八连通区域。八邻接点种子填充算法既能填充四连通区域又能填充八连通区域。

4.4 区域填充算法

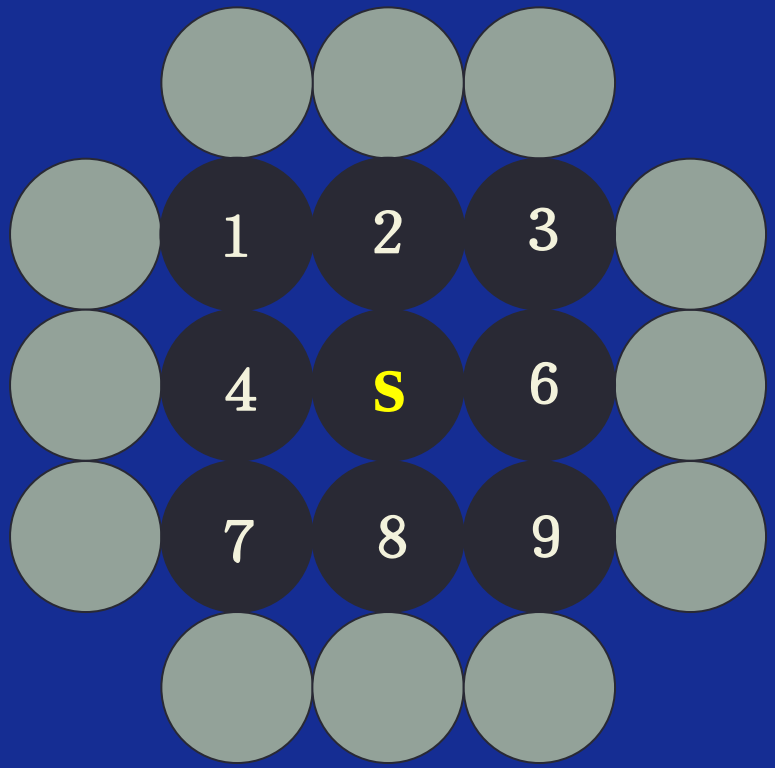
- 区域填充将指定的颜色从种子点扩展到整个区域的过程。能对具有任意复杂边界的区域进行填充
  - (1)从区域内部一个像素点开始,称为种子点(该点已知)
  - (2)判断这个像素是否是一个边界像素点或者已经被填充了
  - (3)如果都不是，就把它填充，然后开始设置邻居像素点。

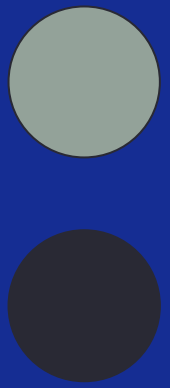


4.4.4 种子填充算法

- 初始化：堆栈置空。将种子点（x，y）入栈。种子像素为栈底像素，若栈不为空，执行如下步骤：
- 出栈：若栈空则结束。否则取栈顶元素；
- 按填充色绘制出栈像素；
- 按左、上、右、下（或左、左上、上、右上、右、右下、下、左下）顺序搜索与出栈像素相邻的4（或8）个像素，若该像素的颜色不是边界色并且未置成填充色，则把该像素入栈；否则丢弃该像素。

种子填充法



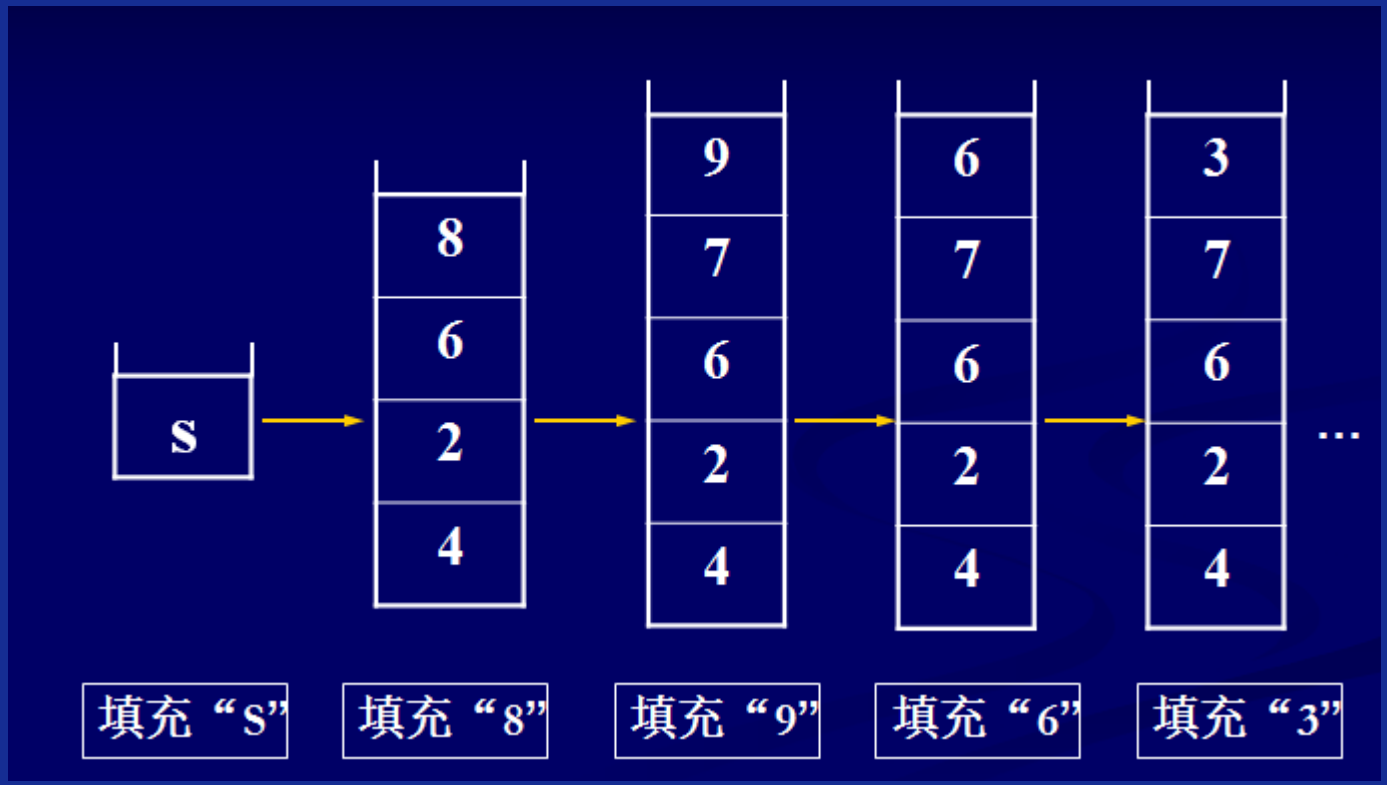


边界点

种子点

种子填充法

堆栈的
变化



区域填充算法(种子填充法)

■缺点：

(1) 有些像素会入栈多次，降低算法效率；栈结构占空间。

(2) 递归执行，算法简单，但效率不高，区域内每一象素都引起一次递归，进/出栈，费时费内存。

■改进算法，减少递归次数，提高效率。

方法之一使用扫描线填充算法；

4.4.5 扫描线种子填充算法

- 目标：减少递归层次
- 适用于内点表示的4连通区域
- 基本过程：

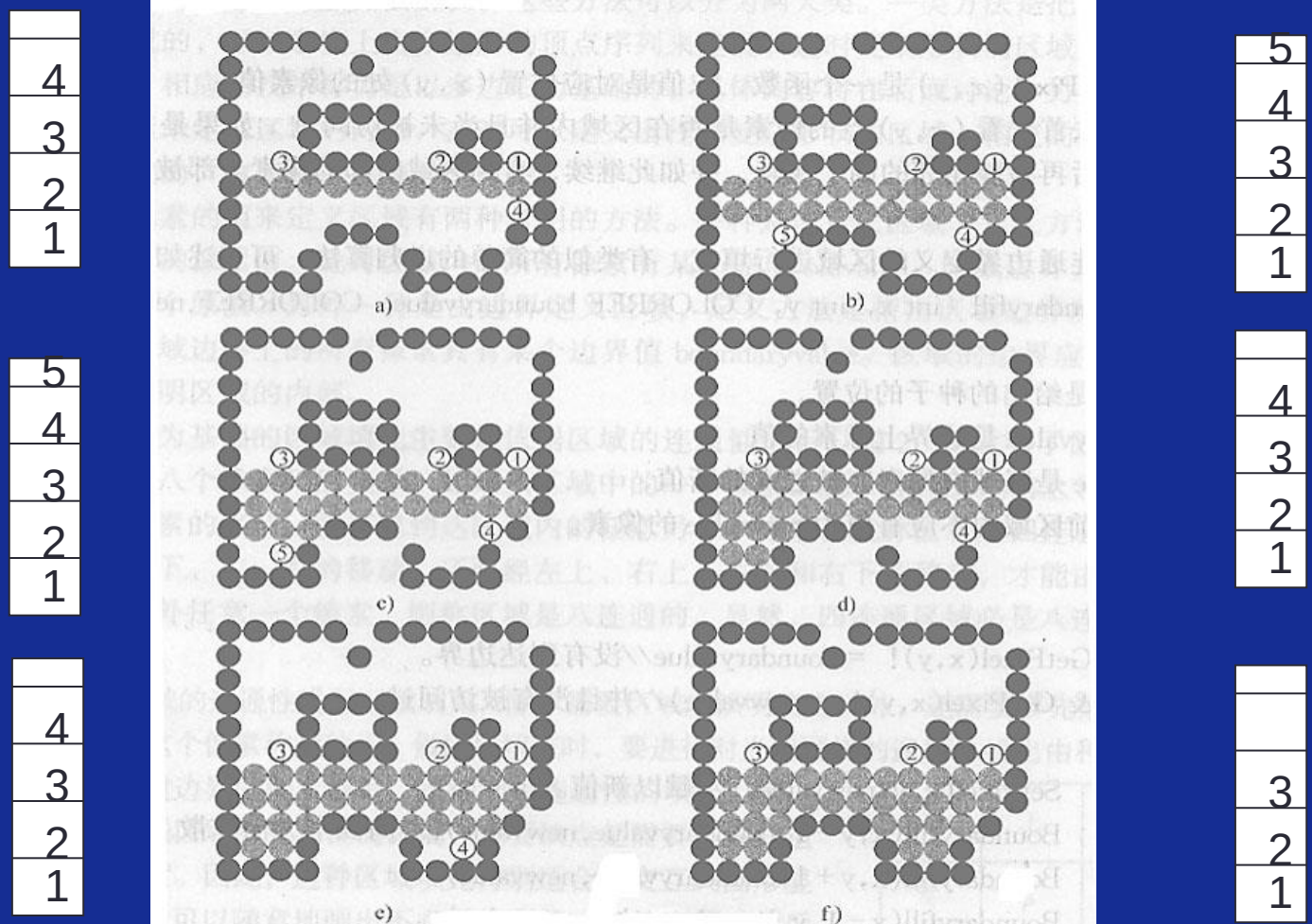
当给定种子点时，首先填充种子点所在的扫描线上的位于给定区域的一个区段，然后确定与这一区段相通的上下两条扫描线上位于给定区域内的区段，并依次保存下来。反复这个过程，直到填充结束。

4.4.5 扫描线种子填充算法

■原理：

- (1)初始化：堆栈置空。将种子点(x，y)入栈；
- (2)出栈：若栈空则结束。否则取栈顶元素(x，y)，以y作为当前扫描线；
- (3)填充并确定种子点所在区段：从种子点(x，y)出发，沿当前扫描线向左、右两个方向填充，直到边界。分别标记区段的左、右端点坐标为x_{left}和x_{right}；
- (4)并确定新的种子点：在区间$[x_{left}, x_{right}]$中检查与当前扫描线y上、下相邻的两条扫描线上的像素。若存在非边界、未填充的像素，则把每一区间的最右像素作为种子点压入堆栈，返回第(2)步。

4.4.5 扫描线种子填充算法



本章小结

- 场景中的物体主要分为多面体与光滑物体。多面体的每个表面就是多边形；光滑物体的表面是多边形网格的集合，掌握好多边形填充算法是绘制真实感图形的基础。
- 有效边表算法、区域填充算法和扫描线种子填充算法
- 有效边表表示的是扫描线在一条边上的连贯性，边表表示的是新边在扫描线上的出现的位置。
- 区域填充算法主要包括四邻接点种子填充算法和八邻接点种子填充算法
- 扫描线种子填充算法是区域填充算法的改进

习题4

1、试写出图4-33所示多边形的边表和扫描线y = 4的有效边表。